

CCCCCCCCCCCC	000000000	NNN	NNN	VVV
CCCCCCCCCCCC	000000000	NNN	NNN	VVV
CCCCCCCCCCCC	000000000	NNN	NNN	VVV
CCC	000	000	NNN	VVV
CCC	000	000	NNN	VVV
CCC	000	000	NNN	VVV
CCC	000	000	NNNNNN	VVV
CCC	000	000	NNNNNN	VVV
CCC	000	000	NNNNNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCC	000	000	NNN NNN	VVV
CCCCCCCCCCCC	000000000	NNN	NNN	VVV
CCCCCCCCCCCC	000000000	NNN	NNN	VVV
CCCCCCCCCCCC	000000000	NNN	NNN	VVV

\*\*FILE\*\*ID\*\*CONVFSTLD

C 11

```
1 0001 0 XTITLE 'VAX-11 CONVERT'  
2 0002 0 MODULE CONVFSTLD ( IDENT='V04-000',  
3 0003 0 OPTLEVEL=3  
4 0004 0 ) =  
5 0005 0  
6 0006 1 BEGIN  
7 0007 1  
8 0008 1 *****  
9 0009 1 *  
10 0010 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY  
11 0011 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.  
12 0012 1 * ALL RIGHTS RESERVED.  
13 0013 1 *  
14 0014 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  
15 0015 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE  
16 0016 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER  
17 0017 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  
18 0018 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY  
19 0019 1 * TRANSFERRED.  
20 0020 1 *  
21 0021 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE  
22 0022 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT  
23 0023 1 * CORPORATION.  
24 0024 1 *  
25 0025 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS  
26 0026 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.  
27 0027 1 *  
28 0028 1 *  
29 0029 1 *****
```

31 0030 1 ++  
32 0031 1 Facility: VAX-11 CONVERT  
33 0032 1 Abstract: This module contains the high level calls for the fast load  
34 0033 1 process along with the declaratons for the data specifically  
35 0034 1 used by fast load  
36 0035 1  
37 0036 1  
38 0037 1  
39 0038 1 Contents:  
40 0039 1 FAST\_LOAD  
41 0040 1 INIT\_FAST\_LOAD  
42 0041 1 LOAD\_PRIMARY  
43 0042 1 LOAD\_SECONDARY  
44 0043 1 LOAD\_DATA\_BUCKET  
45 0044 1 LOAD\_INDEX\_BUCKET  
46 0045 1 FINISH\_INDEX  
47 0046 1 BACKUP\_INDEX  
48 0047 1  
49 0048 1 Environment:  
50 0049 1  
51 0050 1 VAX/VMS Operating System  
52 0051 1  
53 0052 1 --  
54 0053 1  
55 0054 1  
56 0055 1 Author: Keith B Thompson Creation date: August-1980  
57 0056 1  
58 0057 1  
59 0058 1 Modified by:  
60 0059 1  
61 0060 1 V03-013 RAS0305 Ron Schaefer 7-May-1984  
62 0061 1 Fix check for maximum index level so that we report  
63 0062 1 an error rather than get an access violation if the  
64 0063 1 index level exceeds 31.  
65 0064 1  
66 0065 1 V03-012 JWT0177 Jim Teague 17-Apr-1984  
67 0066 1 CONVERT always tried to load a sidr bucket, even if  
68 0067 1 all records in the file had null keys for the  
69 0068 1 index, thereby corrupting the file. Correct this  
70 0069 1 error by making sure at least one non-null key is  
71 0070 1 encountered for an index before allocating and  
72 0071 1 loading a SIDR bucket.  
73 0072 1  
74 0073 1 V03-011 JWT0143 Jim Teague 25-Nov-1983  
75 0074 1 CONVERT used to blindly add records until the fill  
76 0075 1 factor was exceeded. Now, check to see if adding  
77 0076 1 a record will bring us closer to the fill factor.  
78 0077 1 If we're closer before the addition (even though  
79 0078 1 we may be short of the fill factor), then don't  
80 0079 1 add the record.  
81 0080 1  
82 0081 1 V03-010 KBT0476 Keith B. Thompson 29-Jan-1983  
83 0082 1 Add support for the ADD\_KEY function  
84 0083 1  
85 0084 1 V03-009 KBT0459 Keith B. Thompson 10-Jan-1983  
86 0085 1 Fix a bug when loading p3 sidrs with no dups  
87 0086 1

88	0087	1	V03-008 KBT0404	Keith B. Thompson	19-Nov-1982
89	0088	1	Fix some of the sidr code		
90	0089	1			
91	0090	1	V03-007 KBT0382	Keith B. Thompson	26-Oct-1982
92	0091	1	Add prologue 3 sidr support		
93	0092	1			
94	0093	1	V03-006 KBT0375	Keith B. Thompson	20-Oct-1982
95	0094	1	Check for keys out of order from split_data		
96	0095	1			
97	0096	1	V03-005 KBT0349	Keith B. Thompson	4-Oct-1982
98	0097	1	Use new linkage definitions		
99	0098	1			
100	0099	1	V03-004 KBT0050	Keith Thompson	10-May-1982
101	0100	1	Check for empty file before calling finish index		
102	0101	1			
103	0102	1	V03-003 KBT0047	Keith Thompson	14-Apr-1982
104	0103	1	Fix end condition problem with the index buckets		
105	0104	1			
106	0105	1	V03-002 KBT0022	Keith Thompson	24-Mar-1982
107	0106	1	Fix problem with last data bucket being continuation bucket		
108	0107	1	and more duplicate problems. Change some linkages.		
109	0108	1			
110	0109	1	V03-001 KBT0012	Keith Thompson	16-Mar-1982
111	0110	1	Fix some prologue 3 duplicate bugs in load_data_bucket		
112	0111	1	and remove prologue 3 secondary key code		
113	0112	1	*****		

```

115      0113 1
116      0114 1 PSECT
117      0115 1   OWN    = _CONV$FAST_D (PIC),
118      0116 1   GLOBAL = _CONV$FAST_D (PIC),
119      0117 1   PLIT   = _CONV$PLIT_ (SHARE,PIC),
120      0118 1   CODE   = _CONV$FAST_S (SHARE,PIC);
121      0119 1
122      0120 1 LIBRARY 'SYSSLIBRARY:LIB.L32';
123      0121 1 LIBRARY 'SRC$:CONVERT';
124      0122 1
125      0123 1 DEFINE_ERROR_CODES;
126      0124 1
127      0125 1 EXTERNAL ROUTINE
128      0126 1   CONV$GET_VM          : CL$GET_VM,
129      0127 1   CONV$GET_TEMP_VM   : CL$GET_TEMP_VM,
130      0128 1   CONV$FREE_TEMP_VM  : CL$FREE_TEMP_VM      NOVALUE,
131      0129 1   CONV$SEXCEPTION,
132      0130 1   CONV$SEND_OF_FILE   : NOVALUE,
133      0131 1   CONV$SORT_SECONDARY : CL$SORT_SECONDARY,
134      0132 1   CONV$GET_RECORD     : CL$GET_RECORD,
135      0133 1   CONV$CHECK_S_DUP   : CL$JSB_REG_9,
136      0134 1   CONV$CHECK_NULL    : CL$JSB_REG_9,
137      0135 1   CONV$SPLIT_DATA    : CL$JSB_REG_9,
138      0136 1   CONV$COMPRESS_KEY   : CL$JSB_REG_9 NOVALUE,
139      0137 1   CONV$COMPRESS_INDEX  : CL$JSB_REG_9 NOVALUE,
140      0138 1   CONV$MAKE_INDEX    : CL$JSB_REG_9 NOVALUE,
141      0139 1   CONV$WRITE_VBN      : CL$JSB_REG_9 NOVALUE,
142      0140 1   CONV$COPY_KEY      : CL$COPY_KEY NOVALUE,
143      0141 1   CONV$WRITE_BUCKET   : CL$JSB_REG_9 NOVALUE,
144      0142 1   CONV$GET_BUCKET    : CL$JSB_REG_9 NOVALUE,
145      0143 1   CONV$INIT_BUCKET   : CL$JSB_REG_9 NOVALUE,
146      0144 1   CONV$CREATE_HIGHKEY : CL$JSB_REG_9 NOVALUE,
147      0145 1   CONV$WRITE_PROLOGUE: NOVALUE,
148      0146 1   CONV$CONVERT_VBN_ID : CL$CONVERT_VBN_ID NOVALUE,
149      0147 1   CONV$SET_KEY_DESC   : CL$SET_KEY_DESC,
150      0148 1   CONV$GET_NEXT_KEY   : CL$GET_NEXT_KEY,
151      0149 1   CONV$WRITE_KEY_DESC: CL$WRITE_KEY_DESC NOVALUE:
152      0150 1
153      0151 1 FORWARD ROUTINE
154      0152 1   CONV$INIT_FAST_LOAD : CL$INIT_FAST_LOAD NOVALUE,
155      0153 1   LOAD_PRIMARY       : CL$JSB_REG_9,
156      0154 1   CONV$LOAD_SECONDARY : CL$LOAD_SECONDARY NOVALUE,
157      0155 1   LOAD_DATA_BUCKET   : CL$JSB_REG_8 NOVALUE,
158      0156 1   LOAD_INDEX_BUCKET  : CL$JSB_REG_9 NOVALUE,
159      0157 1   FINISH_INDEX      : CL$JSB_REG_9 NOVALUE,
160      0158 1   BACKUP_INDEX      : CL$JSB_REG_9 NOVALUE;
161      0159 1
162      0160 1 EXTERNAL
163      0161 1   CONV$GL_FILL        : LONG,
164      0162 1
165      0163 1   CONV$GW_OUT_REC_SIZ : SIGNED WORD,           ! Output Rec. Size
166      0164 1
167      0165 1   CONV$GL_RECORD_COUNT:
168      0166 1   CONV$GL_EXCEPT_COUNT:
169      0167 1   CONV$GL_VALID_COUNT,
170      0168 1
171      0169 1   CONV$GW_MAX_REC_SIZ : WORD,                 ! Aprox. size of record buffer

```

```

172      0170 1    CONV$GL_RFA_BUFFER,
173      0171 1
174      0172 1    CONV$AB_IN_RAB      : $RAB DECL,
175      0173 1    CONV$AB_OUT_XABSUM   : $XABSUM DECL,
176      0174 1    CONV$AB_OUT_FAB     : $FAB DECL,
177      0175 1    CONV$AB_OUT_RAB     : $RAB DECL,
178      0176 1    CONV$AB_RFA_RAB     : $RAB DECL,
179      0177 1
180      0178 1    CONV$GL_EOF_VBN      : LONG,
181      0179 1    CONV$GB PROC_V1       : BYTE,
182      0180 1    CONV$GB PROL_V2       : BYTE,
183      0181 1    CONV$GB PROL_V3       : BYTE,
184      0182 1    CONV$AR PROLOGUE     : REF BLOCK [ ,BYTE ],
185      0183 1    CONV$AR AREA_BLOCK    : REF BLOCKVECTOR [ ,AREASC_BLN,BYTE ];
186      0184 1
187      0185 1    LITERAL
188      0186 1    FALSE = 0;
189      0187 1    TRUE = 1;
190      0188 1
191      0189 1    MACRO
192      0190 1    ! Some needed macros to define the data record for a bucket
193      0191 1    !
194      0192 1    IRCSL_RRV_VBN      = 3,0,32,0%,    ! RRV VBN Pointer
195      0193 1    IRCSL_RRV_VBN3     = 5,0,32,0%,    ! RRV VBN Pointer (Prologue 3)
196      0194 1    IRCSW_VAR_SIZ       = 7,0,16,0%,    Var. Rec. Format Size field
197      0195 1    IRCSL_DUPCOUNT    = 2,0,32,0%,    Duplicate count field
198      0196 1    IRCSW_DUPSZ        = 6,0,16,0%,    Size field when dup. are allowed
199      0197 1    IRCSW_NODUPSZ      = 2,0,16,0%,    Size field when dup. are not allowed
200      0198 1    IRCSW_P3SZ         = 0,0,16,0%,    Size field for prologue 3 files
201      0199 1
202      0200 1    ! These macros make the code look a little better
203      0201 1    !
204      0202 1    BKT$W_VBNFS        = .CONV$GW_VBN_FS_PTR,0,16,0%, ! VBN Freespace Pointer in index level
205      0203 1    BKT$W_VBNFS0       = .CONV$GW_VBN_FS_PTR,0,0,16,0%, ! VBN Freespace Pointer at the data level
206      0204 1    BKT$L_LCBPTR        = .CONV$GW_LCB_PTR,0,32,0%; ! Last Contuation Bucket Pointer
207      0205 1
208      0206 1    ! Data Decl. for Fast Load routines
209      0207 1
210      0208 1    GLOBAL
211      0209 1    CONV$GL_RECORD_PTR   : LONG,           ! Pointer to record bffer
212      0210 1
213      0211 1    CONV$GW_VBN_FS_PTR   : WORD,
214      0212 1    CONV$GW_VBN_FS_PTR0  : WORD,
215      0213 1    CONV$GW_LCB_PTR     : WORD,
216      0214 1
217      0215 1    CONV$GL_CTX_BLOCK    : LONG,           ! Pointer to the contex block
218      0216 1    CONV$GL_DUP_BUF      : LONG;           ! Pointer to the Duplicate buffer
219      0217 1
220      0218 1    OWN
221      0219 1    CONTINUATION : BYTE,           ! Continuation bucket
222      0220 1    DUPLICATE   : BYTE SIGNED,    ! Duplicate record
223      0221 1
224      0222 1    SAVE_FREESPACE : WORD,           ! Save pointer for backing up index
225      0223 1    SAVE_KEYFRESPC : WORD,
226      0224 1    SAVE_VBNFS   : WORD;           "
227      0225 1

```

```
229      0226 1 %SBTTL 'FAST_LOAD'  
230      0227 1 GLOBAL ROUTINE CONV$FAST_LOAD : CL$JSB_REG_11 =  
231      0228 1 ++  
232      0229 1  
233      0230 1 Functional Description:  
234      0231 1  
235      0232 1     FAST_LOAD is the driving routine for the fast loading process. It  
236      0233 1     will load the primary key then sort and load all secondary keys if  
237      0234 1     any.  
238      0235 1  
239      0236 1 Calling Sequence:  
240      0237 1     CONV$FAST_LOAD()  
241      0238 1  
242      0239 1 Input Parameters:  
243      0240 1     none  
244      0241 1  
245      0242 1 Implicit Inputs:  
246      0243 1     none  
247      0244 1  
248      0245 1 Output Parameters:  
249      0246 1     none  
250      0247 1  
251      0248 1 Implicit Outputs:  
252      0249 1     none  
253      0250 1  
254      0251 1  
255      0252 1 Routine Value:  
256      0253 1     RMSS_EOF or error code  
257      0254 1  
258      0255 1  
259      0256 1 Routines called:  
260      0257 1  
261      0258 1     CONV$INIT_FAST_LOAD  
262      0259 1     LOAD_PRIMARY  
263      0260 1     CONV$SEND_OF_FILE  
264      0261 1     CONV$WRITE_PROLOGUE  
265      0262 1     CONV$SET_KEY_DESC  
266      0263 1     CONV$SORT_SECONDARY  
267      0264 1     CONV$LOAD_SECONDARY  
268      0265 1     CONV$WRITE_KEY_DESC  
269      0266 1  
270      0267 1 Side Effects:  
271      0268 1     none  
272      0269 1  
273      0270 1  
274      0271 1 --  
275      0272 2 BEGIN  
276      0273 2  
277      0274 2     DEFINE_KEY_DESC;  
278      0275 2     DEFINE_CTX_GLOBAL;  
279      0276 2     DEFINE_BUCKET_GLOBAL;  
280      0277 2  
281      0278 2     ! Init the fast load process for all keys  
282      0279 2  
283      0280 2     CONV$INIT_FAST_LOAD( 0 );  
284      0281 2  
285      0282 2     ! Load the primary data and index
```

```
: 286      0283 2      !  
287      0284 2      RET_ON_ERROR( LOAD_PRIMARY() );  
288      0285 2      ! Write prologue  
289      0286 2      CONV$$WRITE_PROLOGUE();  
290      0287 2      ! Also write the key desc.  
291      0288 2      CONV$$WRITE_KEY_DESC();  
292      0289 2      ! Finish off the input file  
293      0290 2      CONV$$END_OF_FILE();  
294      0291 2      ! Free the space taken up by the loading  
295      0292 2      CONV$$FREE_TEMP_VM();  
296      0293 2      ! Process the secondary keys if there were records put into the  
297      0294 2      output file.  
298      0295 2      NOTE: This could cause secondary key indexes to be uninitialized.  
299      0296 2      At the moment RMS doesn't mind, if they ever do, something must be fixed.  
300      0297 2      IF .CONV$GL_VALID_COUNT GTRU 0  
301      0298 2      THEN  
302      0299 2      ! Loop for each key  
303      0300 2      WHILE CONV$$GET_NEXT_KEY()  
304      0301 2      DO  
305      0302 2      BEGIN  
306      0303 2      ! Set up the sort for the secondary key. The sort is a INDEX sort.  
307      0304 2      This type of sort will produce a file of RFA's and keys of the  
308      0305 2      primary data level we just made.  
309      0306 2      RET_ON_ERROR( CONV$$SORT_SECONDARY() );  
310      0307 2      ! Now that the file is sorted get the data and load it in.  
311      0308 2      CONV$$LOAD_SECONDARY();  
312      0309 2      ! Write the prologue  
313      0310 2      CONV$$WRITE_PROLOGUE();  
314      0311 2      ! And the key descriptor  
315      0312 2      CONV$$WRITE_KEY_DESC();  
316      0313 2      ! Free the space taken up by the last key load  
317      0314 2      CONV$$FREE_TEMP_VM()  
318      0315 2      !  
319      0316 2      END:
```

CONV\$FSTLD  
V04-000

VAX-11 CONVERT  
FAST\_LOAD

: 343 0340 2  
: 344 0341 2 RETURN RMSS\_EOF  
: 345 0342 2  
: 346 0343 1 END;

K 11  
15-Sep-1984 23:49:35 VAX-11 Bliss-32 v4.0-742  
14-Sep-1984 12:14:00 [CONV.SRC]CONVFSTLD.B32;1

Page 8  
(4)

.TITLE CONV\$FSTLD VAX-11 CONVERT  
.IDENT \V04-000\  
.PSECT \_CONV\$FAST\_D,NOEXE, PIC,2  
  
00000 CONV\$GL\_RECORD\_PTR::  
.BLKB 4  
00004 CONV\$GW\_VBN\_FS\_PTR::  
.BLRB 2  
00006 CONV\$GW\_VBN\_FS\_PTR0::  
.BLRB 2  
00008 CONV\$GW\_LCB\_PTR::  
.BLRB 2  
0000A .BLKB 2  
0000C CONV\$GL\_CTX\_BLOCK::  
.BLRB 4  
00010 CONV\$GL\_DUP\_BUF::  
.BLRB 4  
00014 CONTINUATION:  
.BLKB 1  
00015 DUPLICATE:  
.BLKB 1  
00016 SAVE\_FREESPACE:  
.BLKB 2  
00018 SAVE\_KEYFRESPC:  
.BLKB 2  
0001A SAVE\_VBNFS:  
.BLKB 2  
  
.EXTRN CONVERT\$\_FACILITY  
.EXTRN CONVS\_FA0\_MAX, CONVS\_BADBLK  
.EXTRN CONVS\_BADLOGIC, CONVS\_BADSORT  
.EXTRN CONVS\_CONFQUAL, CONVS\_CREATEDSTM  
.EXTRN CONVS\_CREA\_ERR, CONVS\_DELPRI  
.EXTRN CONVS\_DUP, CONVS\_EXTN\_ERR  
.EXTRN CONVS\_FATALEXC, CONVS\_FILLIM  
.EXTRN CONVS\_IDX\_LIM, CONVS\_ILL\_KEY  
.EXTRN CONVS\_ILL\_VALUE  
.EXTRN CONVS\_INP\_FILES  
.EXTRN CONVS\_INSVRMEM  
.EXTRN CONVS\_INVBKT, CONVS\_KEY  
.EXTRN CONVS\_KEYREF, CONVS\_LOADIDX  
.EXTRN CONVS\_NARG, CONVS\_NI  
.EXTRN CONVS\_NOKEY, CONVS\_NOTIDX  
.EXTRN CONVS\_NOTSEQ, CONVS\_NOWILD  
.EXTRN CONVS\_ORDER, CONVS\_OPENEXEC  
.EXTRN CONVS\_OPENIN, CONVS\_OPENOUT  
.EXTRN CONVS\_PAD, CONVS\_PLV  
.EXTRN CONVS\_PROERR, CONVS\_PROL\_WRT  
.EXTRN CONVS\_READERR, CONVS\_RSK  
.EXTRN CONVS\_RSZ, CONVS\_RTL

.EXTRN CONVS\_RTS, CONVS\_SEQ  
 .EXTRN CONVS\_UDF\_BKS, CONVS\_UDF\_BLK  
 .EXTRN CONVS\_VFC, CONVS\_WRITEERR  
 .EXTRN CONVSSGET\_VM, CONVSSGET\_TEMP\_VM  
 .EXTRN CONVSSFREE\_TEMP\_VM  
 .EXTRN CONVSSCEPTION  
 .EXTRN CONVSSEND\_OF\_FILE  
 .EXTRN CONVSSORT\_SECONDARY  
 .EXTRN CONVSSGET\_RECORD  
 .EXTRN CONVSSCHECK\_S\_DUP  
 .EXTRN CONVSSCHECK\_NULL  
 .EXTRN CONVSSPLIT\_DATA  
 .EXTRN CONVSSCOMPRESS\_KEY  
 .EXTRN CONVSSCOMPRESS\_INDEX  
 .EXTRN CONVSSMAKE\_INDEX  
 .EXTRN CONVSSWRITE\_VBN  
 .EXTRN CONVSSCOPYKEY, CONVSSWRITE\_BUCKET  
 .EXTRN CONVSSGET\_BUCKET  
 .EXTRN CONVSSINIT\_BUCKET  
 .EXTRN CONVSSCREATE\_HIGH\_KEY  
 .EXTRN CONVSSWRITE\_PROLOGUE  
 .EXTRN CONVSSCONVERT\_VBN\_ID  
 .EXTRN CONVSSSET\_KEY\_DESC  
 .EXTRN CONVSSGET\_NEXT\_KEY  
 .EXTRN CONVSSWRITE\_KEY\_DESC  
 .EXTRN CONVSGL\_FILE, CONVSGW\_OUT\_REC\_SIZ  
 .EXTRN CONVSGL\_RECORD\_COUNT  
 .EXTRN CONVSGL\_EXCEPT\_COUNT  
 .EXTRN CONVSGL\_VALID\_COUNT  
 .EXTRN CONVSGW\_MAX\_REC\_SIZ  
 .EXTRN CONVSGL\_RFA\_BUFFER  
 .EXTRN CONVSAB\_IN\_RAB, CONVSAB\_OUT\_XABSUM  
 .EXTRN CONVSAB\_OUT\_FAB  
 .EXTRN CONVSAB\_OUT\_RAB  
 .EXTRN CONVSAB\_RFA\_RAB  
 .EXTRN CONVSGL\_EOF\_VBN  
 .EXTRN CONVSGB\_PROL\_V1  
 .EXTRN CONVSGB\_PROL\_V2  
 .EXTRN CONVSGB\_PROL\_V3  
 .EXTRN CONVSAR\_PROLOGUE  
 .EXTRN CONVSAR\_AREA\_BLOCK

.PSECT \_CONV\$FAST\_S,NOWRT, SHR, PIC,2

			59 7D 00000 CONVSSFAST LOAD:		
			MOVQ R9, -(SP)		0227
			CLRL -(SP)		0280
			BSBW CONVSSINIT_FAST_LOAD		
			ADDL2 #4, SP		
			BSBW LOAD_PRIMARY		0284
			BLBC STATUS, 3S		
			CALLS #0, CONVSSWRITE_PROLOGUE		0288
			CONVSSWRITE_KEY_DESC		0292
			CALLS #0, CONVSSEND_OF_FILE		0296
			CONVSSFREE_TEMP_VM		0300
			TSTL CONVSGL_VALID_COUNT		0308
			BEQL 2S		

CONV\$FSTLD  
V04-000

VAX-11 CONVERT  
FAST\_LOAD

M 11

15-Sep-1984 23:49:35  
14-Sep-1984 12:14:00

VAX-11 Bliss-32 V4.0-742  
[CONV.SRC]CONVFSTLD.B32;1

Page 10  
(4)

16	0000G 30 00027 1\$:	BSBW	CONV\$GET_NEXT_KEY	: 0313
	50 F9 0002A	BLBC	R0, 2\$	
17	0000G 30 0002D	BSBW	CONV\$SORT_SECONDARY	: 0321
	50 E9 00030	BLBC	STATUS, 3\$	
0000G CF	0000V 30 00033	BSBW	CONV\$LOAD_SECONDARY	: 0325
	00 FB 00036	CALLS	#0, CONV\$WRITE_PROLOGUE	: 0329
	0000G 30 0003B	BSBW	CONV\$WRITE_KEY_DESC	: 0333
	0000G 30 0003E	BSBW	CONV\$FREE_TEMP_VM	: 0337
	E4 11 00041	BRB	1\$	
50 0001827A	8F D0 00043 2\$:	MOVL	#98938, R0	: 0341
59	8E 7D 0004A 3\$:	MOVQ	(SP)+, R9	
	05 0004D	RSB		: 0343

: Routine Size: 78 bytes, Routine Base: \_CONV\$FAST\_S + 0000

```
348 0344 1 %SBTTL 'INIT_FAST_LOAD'
349 0345 1 GLOBAL ROUTINE CONV$INIT_FAST_LOAD ( MAX_KEY ) : CL$INIT_FAST_LOAD NOVALUE =
350 0346 1 ++
351 0347 1 Functional Description:
352 0348 1
353 0349 1 Initialize the fast load process. Get memory for buffers and set up
354 0350 1 pointers. There are up to 3 pointers to record buffers at each level
355 0351 1 RCP, RDP and LKB for level 0 and prologue 3 files. The RCP, LKP and
356 0352 1 RDP for all but level 0 pointers are set here. The proper sizes are:
357 0353 1
358 0354 1
359 0355 1
360 0356 1 LEVEL 0 RCP --->: max_key + 13
361 0357 1
362 0358 1
363 0359 1
364 0360 1 LEVEL 1+ RCP --->: 5
365 0361 1
366 0362 1
367 0363 1
368 0364 1 RDP --->: max_key + 2
369 0365 1
370 0366 1
371 0367 1
372 0368 1 ALL LEVELS LKP --->: max_key
373 0369 1
374 0370 1
375 0371 1 The RDP for level 0 is set in load_primary and load_secondary.
376 0372 1
377 0373 1 Calling Sequence:
378 0374 1
379 0375 1 INIT_FAST_LOAD();
380 0376 1
381 0377 1 Input Parameters:
382 0378 1 none
383 0379 1
384 0380 1 Implicit Inputs:
385 0381 1 none
386 0382 1
387 0383 1 Output Parameters:
388 0384 1 none
389 0385 1
390 0386 1 Implicit Outputs:
391 0387 1 none
392 0388 1
393 0389 1 Routine Value:
394 0390 1 none
395 0391 1
396 0392 1 Routines Called:
397 0393 1
398 0394 1 CONV$SET_KEY_DESC
399 0395 1 CONV$GET_NEXT_KEY
400 0396 1 CONV$GET_VBN
401 0397 1
402 0398 1 Side Effects:
403 0399 1
404 0400 1 Sets the end of file VBN pointer. Allocates memory for record buffers.
```

```
405      0401 1 | Sets up the record data pointers, record control pointers and last key
406      0402 1 |
407      0403 1 |
408      0404 1 |--|
409      0405 1 |
410      0406 2 BEGIN
411      0407 2
412      0408 2 DEFINE_CTX;
413      0409 2 DEFINE_BUCKET;
414      0410 2 DEFINE_KEY_DESC;
415      0411 2
416      0412 2 ! Since we are doing block IO we dont need the XABs anymore
417      0413 2 ! If they are keep around area xabs (if any) will override the fab during
418      0414 2 an extend a screw everything up.
419      0415 2
420      0416 2 CONVSAB_OUT_FAB [ FAB$L_XAB ] = 0;
421      0417 2
422      0418 2 ! Find the end of file VBN. In a new file it the one block past the last
423      0419 2 allocated area (the last area may not be allocated therefore look at
424      0420 2 one)
425      0421 2
426      0422 2 BEGIN ! HIGH_VBN local
427      0423 2
428      0424 3 LOCAL HIGH_VBN;
429      0425 3
430      0426 3 HIGH_VBN = 0;
431      0427 3
432      0428 3 INCR AREA FROM 0 TO ( .CONVSAB_OUT_XABSUM [ XAB$B_NOA ] - 1 ) BY 1
433      0429 3 DO
434      0430 3
435      0431 3 ! If the current extent starts at a higher VBN then the last one
436      0432 3 us this one to find the end of file
437      0433 3
438      0434 3 IF .CONVSAR_AREA_BLOCK [ .AREA,AREASL_CVBN ] GTR .HIGH_VBN
439      0435 3 THEN
440      0436 3
441      0437 3 ! The end of file is this the start of this extent plus the number
442      0438 3 of blocks in the extent
443      0439 3
444      0440 3 CONV$GL_EOF_VBN = .CONVSAR_AREA_BLOCK [ .AREA,AREASL_CVBN ] +
445      0441 3 .CONVSAR_AREA_BLOCK [ .AREA,AREASL_CNBLK ];
446      0442 3
447      0443 2 END; ! HIGH_VBN local
448      0444 2
449      0445 2 ! Calculate the max space needed for index key buffers and init. the context
450      0446 2 block. If it was not given.
451      0447 2
452      0448 2 IF .MAX_KEY EQLU 0
453      0449 2 THEN
454      0450 3 BEGIN
455      0451 3
456      0452 3 ! The max. size is the size of the longest key.
457      0453 3 So we check each key desc.
458      0454 3
459      0455 3 CONV$SET_KEY_DESC( 0 );
460      0456 3
461      0457 3 DO
```

```
; 462      0458 3
; 463      0459
; 464      0460      IF .KEY_DESC [ KEYSB_KEYSZ ] GTR .MAX_KEY
; 465      0461      THEN      MAX_KEY = .KEY_DESC [ KEYSB_KEYSZ ]
; 466      0462
; 467      0463      UNTIL NOT CONV$GET_NEXT_KEY()
; 468      0464      END;
; 469      0465
; 470      0466      BEGIN
; 471      0467      LOCAL      BYTES;
; 472      0468      ! Figure the total number of bytes. (SEE ABOVE)
; 473      0469      BYTES = ( .MAX_KEY * ( MAX_IDX_LVL + 1 ) ) + ( MAX_IDX_LVL * 7 ) + 13;
; 474      0470      ! For Prologue 3 files we may need the last key buffers
; 475      0471      IF .CONV$GB_PROL_V3
; 476      0472      THEN      BYTES = .BYTES + ( .MAX_KEY * ( MAX_IDX_LVL - 1 ) );
; 477      0473
; 478      0474      ! Add the space for the context block
; 479      0475      BYTES = .BYTES + ( MAX_IDX_LVL * CTX$K_BLN );
; 480      0476
; 481      0477      ! Get the zero filled space
; 482      0478      CONV$GL_CTX_BLOCK = CONV$GET_VM ( .BYTES )
; 483      0479
; 484      0480      END;
; 485      0481
; 486      0482      ! Set all of the record control pointers and record data pointers for
; 487      0483      ! level one (1) and above.
; 488      0484      CTX = .CONV$GL_CTX_BLOCK;
; 489      0485
; 490      0486      CTX [ CTX$L_RCP ] = .CTX + ( MAX_IDX_LVL * CTX$K_BLN );
; 491      0487
; 492      0488      BEGIN      ! BUFFER_OFFSET local
; 493      0489
; 494      0490      LOCAL      BUFFER_OFFSET;
; 495      0491      BUFFER_OFFSET = .CTX [ CTX$L_RCP ] + .MAX_KEY + 13;
; 496      0492
; 497      0493      INCR I FROM 1 TO ( MAX_IDX_LVL - 1 ) BY 1
; 498      0494      DO
; 499      0495      BEGIN
; 500      0496      CTX = .CTX + CTX$K_BLN;
; 501      0497      CTX [ CTX$B_LEVEL ] = .I;
; 502      0498      CTX [ CTX$L_RCP ] = .BUFFER_OFFSET;
; 503      0499      CTX [ CTX$L_RDP ] = .BUFFER_OFFSET + 5;
; 504      0500      BUFFER_OFFSET = .BUFFER_OFFSET + .MAX_KEY + 7
; 505      0501
; 506      0502      END;
; 507      0503
; 508      0504      ! Set up the last key buffer for level 0
; 509      0505
; 510      0506
; 511      0507
; 512      0508
; 513      0509
; 514      0510
; 515      0511
; 516      0512
; 517      0513
; 518      0514
```

```

: 519      0515 3    CTX = .CONV$GL_CTX_BLOCK;
: 520      0516 3    CTX [ CTXSL_LKP ] = .BUFFER_OFFSET;
: 521      0517      ! Set up the last key buffer if necessary for levels 1 and above
: 522      0518
: 523      0519
: 524      0520      IF .CONV$GB_PROL_V3
: 525      0521      THEN
: 526      0522      INCR I FROM 1 TO ( MAX_IDX_LVL - 1 ) BY 1
: 527      0523      DO
: 528      0524      BEGIN
: 529      0525      CTX = .CTX + CTX$K_BLN;
: 530      0526      BUFFER_OFFSET = BUFFER_OFFSET + MAX_KEY;
: 531      0527      CTX [ CTXSL_LKP ] = .BUFFER_OFFSET
: 532      0528      END
: 533      0529
: 534      0530      END;      ! BUFFER_OFFSET local
: 535      0531
: 536      0532      RETURN
: 537      0533
: 538      0534      END;

```

1C BB 00000 CONV\$INIT_FAST_LOAD::									
						PUSHR	#^M<R2,R3,R4>		0345
				0000G	CF D4 00002	CLRL	CONV\$AB_OUT_FAB+36		0416
				53 50	54 D4 00006	CLRL	HIGH_VBN		0426
				0000G	CF 9A 00008	MOVZBL	CONV\$AB_OUT_XABSUM+8, R3		0428
				01	CE 0000D	MNEGL	#1, AREA		0434
				1D	11 00010	BRB	2\$		
				52 51	06 78 00012	1\$:	ASHL	#6, AREA, R2	
				52 54	CF C1 00016	ADDL3	CONV\$AR_AREA_BLOCK, R2, R1		
				OC	A1 D1 0001C	CMPL	12(R1), HIGH_VBN		
				0000G	OD 15 00020	BLEQ	2\$		
				52	CF C0 00022	ADDL2	CONV\$AR_AREA_BLOCK, R2		0441
				51	10 A2 C1 00027	ADDL3	16(R2), 12(R1), CONV\$GL_EOF_VBN		
				52	53 F2 0002F	AOBLSS	R3, AREA, 1\$		0434
				51	2\$:	TSTL	MAX_KEY		0448
				50	10 AE D5 00033	BNEQ	5\$		
					1C 12 00036	CLRL	- (SP)		0455
					7E D4 00038	BSBW	CONV\$SET_KEY_DESC		
					0000G 30 0003A	ADDL2	#4, SP		
					04 C0 0003D	CMPZV	#0, #8, 20(KEY_DESC), MAX_KEY		0459
					00 ED 00040	3\$:			
					05 15 00047	BLEQ	4\$		
					14 AB 9A 00049	MOVZBL	20(KEY_DESC), MAX_KEY		0461
					0000G 30 0004E	BSBW	CONV\$GET_NEXT_KEY		0463
					50 E8 00051	BLBS	R0, 3\$		
					52 10 AE D0 00054	MOVL	MAX_KEY, R2		0472
					21 C5 00058	MULL3	#33, R2, R1		
					50 00ED C1 9E 0005C	MOVAB	237(R1), BYTES		
					07 0000G CF E9 00061	BLBC	CONV\$GB_PROL_V3, 6\$		0476
					52 1F C5 00066	MULL3	#31, R2, R1		0478
					50 0B80 C0 9E 0006D	ADDL2	R1, BYTES		
					6\$:	MOVAB	2944(R0), BYTES		0482
					50 DD 00072	PUSHL	BYTES		0486

		0000G	30	00074	BSBW	CONV\$GET_VM	
		04	C0	00077	ADDL2	#4, SP	
		50	DO	0007A	MOVL	R0, CONV\$GL_CTX_BLOCK	
		5A	CF	0007F	MOVL	CONV\$GL_CTX_BLOCK, CTX	
	50	AA	0B80	CA	MOVAB	2944(R10), Z8(CTX)	
		52	30	AA	ADDL3	48(CTX), R2, R0	
		50	AA	C1	ADDL2	#13, BUFFER_OFFSET	
		51	0D	CO	MOVL	#1, I	
		51	01	DO	MOVAB	92(R10), CTX	
	50	5A	5C	AA	51	I, 2(CTX)	
		02	AA	9E	00099	MOVBL	BUFFER_OFFSET, 48(CTX)
		30	AA	50	0009D	MOVAB	5(R0)-52(CTX)
		34	AA	05	A0	MOVAB	7(R2)[BUFFER_OFFSET], BUFFER_OFFSET
	E6	50	07	A240	9E	#31, I, 7\$	
		51	5A	0000'	F3	CONV\$GL_CTX_BLOCK, CTX	
		51	AA	CF	000AF	MOVL	BUFFER_OFFSET, 60(CTX)
		3C	AA	50	DO	BLBC	CONV\$GB_PROL_V3, 9\$
		12	0000G	CF	E9	#1, I	
		51	5A	5C	000B8	MOVAB	92(R10), CTX
		51	01	AA	000BD	ADDL2	R2, BUFFER_OFFSET
		50	5A	9E	000C0	MOVL	BUFFER_OFFSET, 60(CTX)
	F1	3C	AA	52	000C4	AOBLEQ	#31, I, 8\$
		51	50	50	DO	POPR	#^M<R2,R3,R4>
		1C	1F	F3	000CB	RSB	
		05	BA	000CF	9\$:		
				000D1			

; Routine Size: 210 bytes, Routine Base: \_CONV\$FAST\_S + 004E

```
540      0535 1 %SBTTL 'LOAD PRIMARY'  
541      0536 1 ROUTINE LOAD_PRIMARY : CL$JSB_REG_9 =  
542      0537 1 ++  
543      0538 1  
544      0539 1 Functional Description:  
545      0540 1  
546      0541 1 Loads the primary key of a index sequential file.  
547      0542 1  
548      0543 1 Calling Sequence:  
549      0544 1  
550      0545 1 LOAD_PRIMARY()  
551      0546 1  
552      0547 1 Input Parameters:  
553      0548 1 none  
554      0549 1  
555      0550 1 Implicit Inputs:  
556      0551 1 none  
557      0552 1  
558      0553 1 Output Parameters:  
559      0554 1 none  
560      0555 1  
561      0556 1 Implicit Outputs:  
562      0557 1 none  
563      0558 1  
564      0559 1 Routine Value:  
565      0560 1  
566      0561 1 RMSS_EOF or error codes  
567      0562 1  
568      0563 1 Routine Called:  
569      0564 1  
570      0565 1 CONV$$SET_KEY_DESC  
571      0566 1 CONV$$GET_TEMP_VM  
572      0567 1 CONV$$GET_BUCKET  
573      0568 1 CONV$$GET_RECORD  
574      0569 1 CONV$SEXCEPTION  
575      0570 1 CONV$$SPLIT_DATA  
576      0571 1 LOAD DATA BUCKET  
577      0572 1 FINISH_INDEX  
578      0573 1  
579      0574 1 Side Effects:  
580      0575 1  
581      0576 1 Loads primary key  
582      0577 1  
583      0578 1 --  
584      0579 1  
585      0580 2 BEGIN  
586      0581 2  
587      0582 2 LABEL  
588      0583 2 DUP_BLK;  
589      0584 2  
590      0585 2 DEFINE_CTX;  
591      0586 2 DEFINE_BUCKET;  
592      0587 2 DEFINE_KEY_DESC;  
593      0588 2  
594      0589 2 CTX = .CONV$GL_CTX_BLOCK;  
595      0590 2  
596      0591 2 ! Set key to the primary index
```

```
: 597      0592 2      ! CONV$SET_KEY_DESC( 0 );
: 598      0593 2      ! Errors on the rab from now on are WRITEERRs
: 599      0594 2
: 600      0595 2      CONV$AB_OUT_RAB [ RAB$L_CTX ] = CONV$WRITEERR;
: 601      0596 2
: 602      0597 2      ! For prologue 3 files we need an extra buffer for the data record
: 603      0598 2      Else we let the REC_DATA_PTR point to the user buffer of the output rab
: 604      0599 2
: 605      0600 2
: 606      0601 2
: 607      0602 2      IF .CONV$GB_PROL_V3
: 608      0603 2      THEN
: 609      0604 2          BEGIN
: 610      0605 2              LOCAL  BYTES;
: 611      0606 2              ! The worst case is fully non compressed record with compression info
: 612      0607 2              BYTES = .CONV$GW_MAX_REC_SIZ + 3;
: 613      0608 2
: 614      0609 2
: 615      0610 2          ! Get the space for the data buffer
: 616      0611 2
: 617      0612 2          ! Record data pointer at level 0 will point to the new buffer
: 618      0613 2
: 619      0614 2          CTX [ CTX$L_RDP ] = CONV$GET_TEMP_VM ( .BYTES )
: 620      0615 2
: 621      0616 2
: 622      0617 2
: 623      0618 2      END
: 624      0619 2
: 625      0620 2
: 626      0621 2      ! Record data pointer at level 0 points to Record Ptr
: 627      0622 2
: 628      0623 2          CTX [ CTX$L_RDP ] = .CONV$GL_RECORD_PTR;
: 629      0624 2
: 630      0625 2      ! Get the Buckets for the data area and at least the first level of the index
: 631      0626 2
: 632      0627 2      Get the bucket for level 0
: 633      0628 2
: 634      0629 2          CONV$GET_BUCKET( .KEY_DESC [ KEY$B_DANUM ] );
: 635      0630 2
: 636      0631 2          KEY_DESC [ KEY$L_LDVBN ] = .CTX [ CTX$L_CURRENT_VBN ];
: 637      0632 2
: 638      0633 2      ! Get the bucket for level 1
: 639      0634 2
: 640      0635 2          CTX = .CTX + CTX$K_BLN;
: 641      0636 2          CONV$GET_BUCKET( .KEY_DESC [ KEY$B_LANUM ] );
: 642      0637 2
: 643      0638 2          CTX = .CONV$GL_CTX_BLOCK;
: 644      0639 2
: 645      0640 2      ! For the primary key the Data comes from GET_RECORD. NOTE: Don't use the
: 646      0641 2      ! UBF of the input RAB since some record conversion may be done. Also note
: 647      0642 2      ! the RBF pointer of the output RAB is destroyed after the first call to
: 648      0643 2      ! WRITE_BUCKET but it is ok to use it now.
: 649      0644 2
: 650      0645 2
: 651      0646 2
: 652      0647 2      BEGIN
: 653      0648 2          DEFINE_RECORD_CTRL_GLOBAL;
```

```
654      0649 3 LOCAL
655      0650 3 STATUS;
656      0651
657      0652 RECORD_CTRL = .CTX [ CTXSL_RCP ];
658      0653
659      0654 ! Main record processing loop. The call to GET_RECORD does any record format
660      0655 processing and exception handling before it returns. The size of the record
661      0656 is passed back by OUT_REC_SIZ.
662      0657
663      0658 WHILE ( STATUS = CONV$GET_RECORD() )
664      0659 DO
665      0660 BEGIN ! Main Loop
666      0661
667      0662 DUP_BLK:
668      0663 BEGIN ! DUP_BLK Primary duplicate block
669      0664
670      0665 ! If the record is shorter the minium record length of the primary key
671      0666 cause an exception
672      0667
673      0668 IF .CONV$GW_OUT_REC_SIZ LSS .KEY_DESC [ KEY$W_MINRECSZ ]
674      0669 THEN
675      0670 BEGIN
676      0671
677      0672 LOCAL STATUS;
678      0673
679      0674 ! If it was not fatal continue else exit
680      0675
681      0676 IF STATUS = CONV$SEXCEPTION( CONV$_RSK )
682      0677 THEN
683      0678 LEAVE DUP_BLK
684      0679 ELSE
685      0680 RETURN .STATUS
686      0681 END;
687      0682
688      0683 ! Seperate the key from the data record if necessary and do
689      0684 data compression if necessary also check if this is a duplicate
690      0685 or the key is out of order
691      0686
692      0687 DUPLICATE = CONV$SPLIT_DATA();
693      0688
694      0689 ! If out of order, i.e. duplicate = -1, signal exception and continue
695      0690
696      0691 IF .DUPLICATE LSS 0
697      0692 THEN
698      0693 BEGIN
699      0694
700      0695 LOCAL STATUS;
701      0696
702      0697 ! If not fatal exception then continue else bomb
703      0698
704      0699 IF STATUS = CONV$SEXCEPTION ( CONV$_SEQ )
705      0700 THEN
706      0701 LEAVE DUP_BLK
707      0702 ELSE
708      0703 RETURN .STATUS
709      0704
710      0705 END;
```

```
711    0706 5
712    0707 5      ! If we got a dup and we dont allow dups then cause an exception
713    0708 5
714    0709 6      IF .DUPLICATE AND ( NOT .KEY_DESC [ KEY$V_DUPKEYS ] )
715    0710 5      THEN
716    0711 6      BEGIN
717    0712 6
718    0713 6      LOCAL      STATUS;
719    0714 6
720    0715 6      ! If not fatal exception then continue else bomb
721    0716 6
722    0717 6      IF STATUS = CONV$SEXCEPTION ( CONV$_DUP )
723    0718 6      THEN
724    0719 6      LEAVE DUP_BLK
725    0720 6      ELSE
726    0721 6      RETURN .STATUS
727    0722 6
728    0723 5
729    0724 5
730    0725 5      ! Set up the control byte for the record
731    0726 5
732    0727 5      RECORD_CTRL [ IRC$B_CONTROL ] = 2;
733    0728 5
734    0729 5      ! Set the size field int the record
735    0730 5
736    0731 5      IF .CONV$GB_PROL_V3
737    0732 5      THEN
738    0733 6      BEGIN
739    0734 6
740    0735 6      ! A small non compressed fixed length record has no size field
741    0736 6
742    0737 6      IF .KEY_DESC [ KEY$V_REC_COMPR ] OR
743    0738 6      .KEY_DESC [ KEY$V_KEY_COMPR ] OR
744    0739 7      ( .CONV$AB_OUT_FAB [ FAB$B_RFIM ] EQLU FAB$C_VAR )
745    0740 6      THEN
746    0741 6      RECORD_CTRL [ 9,0,16,0 ] = .CTX [ CTX$W_RCS ] +
747    0742 6      .CTX [ CTX$W_RDS ] - 11
748    0743 6
749    0744 5      END
750    0745 5
751    0746 5      ELSE
752    0747 5      ! Set up the record size for var. length records
753    0748 5      ! for prologue 1 and 2 files
754    0749 5
755    0750 5      IF .CONV$AB_OUT_FAB [ FAB$B_RFIM ] EQLU FAB$C_VAR
756    0751 5      THEN
757    0752 5      RECORD_CTRL [ IRC$W_VAR_SIZ ] = .CONV$GW_OUT_REC_SIZ;
758    0753 5
759    0754 5      ! If we are in a continuation bucket and the current record is NOT a
760    0755 5      ! duplicate then write the current bucket out and start a new one
761    0756 5
762    0757 5      ! For optimumimization do the continuation check first
763    0758 5
764    0759 5      IF .CONTINUATION THEN IF NOT .DUPLICATE
765    0760 6      THEN
766    0761 6      BEGIN
767    0762 6      CONV$SWRITE_BUCKET();
```

```

768      0763 6
769      0764 6      CONV$$INIT_BUCKET();
770      0765 6
771      0766 6      ! Continuation no longer need be set. (the next record will always
772      0767 6      fix into the new bucket
773      0768 6
774      0769 6      CONTINUATION = _CLEAR
775      0770 6
776      0771 5
777      0772 5      END;
778      0773 5      ! Load the record
779      0774 5
780      0775 5      LOAD_DATA_BUCKET()
781      0776 5
782      0777 5      END           ! DUP_BLK Primary duplicate block
783      0778 3      END;        ! Main loop
784      0779 3
785      0780 3
786      0781 3      ! If we exited because of end of file and there are records in the file
787      0782 3      then finish off the index
788      0783 3      IF ( .STATUS EQLU RMSS_EOF ) AND
789      0784 4          ( .CONV$GL_RECORD_COUNT NEQU .CONV$GL_EXCEPT_COUNT )
790      0785 3      THEN
791      0786 3          FINISH_INDEX();
792      0787 3
793      0788 3
794      0789 3
795      0790 3      RETURN CONV$_SUCCESS
796      0791 1      END;
END;

```

		0104 8F BB 00000 LOAD_PRIMARY:			
	5A	0000' CF D0 00004	PUSHR	#^M<R2,R8>	0536
		7E D4 00009	MOVL	CONV\$GL_CTX_BLOCK, CTX	0589
		0000G 30 0000B	CLRL	-(SP)	0593
0000G	5E	0000G 04 C0 0000E	BSBW	CONV\$\$SET_KEY_DESC	
	CF	00000000G 8F D0 00011	ADDL2	#4, SP	0597
	16	0000G CF E9 0001A	MOVL	#CONV\$ WRITEERR, CONV\$AB_OUT_RAB+24	0602
	50	0000G CF 3C 0001F	BLBC	CONV\$GB_PROL_V3, 1\$	0610
	50	03 C0 00024	MOVZWL	CONV\$GW_MAX_REC_SIZ, BYTES	
		50 DD 00027	ADDL2	#3, BYTES	0616
		0000G 30 00029	PUSHL	BYTES	
		0000G 04 C0 0002C	BSBW	CONV\$\$GET_TEMP_VM	
34	AA	5E 04 C0 0002C	ADDL2	#4, SP	0623
		50 D0 0002F	MOVL	R0, 52(CTX)	0629
		06 11 00033	BRB	2\$	
34	AA	0000' CF D0 00035	MOVL	CONV\$GL RECORD PTR, 52(CTX)	0631
	7E	08 AB 9A 0003B	MOVZBL	8(KEY DESC), -(SP)	0635
		0000G 30 0003F	BSBW	CONV\$\$GET_BUCKET	0636
54	AB	08 AA D0 00042	MOVL	8(CTX), 84(KEY_DESC)	
	5A	5C AA 9E 00047	MOVAB	92(R10), CTX	
	6E	07 AB 9A 0004B	MOVZBL	7(KEY DESC), (SP)	
		0000G 30 0004F	BSBW	CONV\$\$GET_BUCKET	
	5E	04 C0 00052	ADDL2	#4, SP	

CONV\$F STLD  
V04-000

VAX-11 CONVERT  
LOAD PRIMARY

K 12

15-Sep-1984 23:49:35  
14-Sep-1984 12:14:00

VAX-11 Bliss-32 V4.0-742  
[CONV.SRC]CONVFSTLD.B32:1

Page 21  
(6)

; Routine Size: 280 bytes, Routine Base: \_CONV\$FAST\_S + 0120

; 797 0792 1

```
799      0793 1 %SBTTL 'LOAD SECONDARY'
800      0794 1 GLOBAL ROUTINE CONV$LOAD_SECONDARY : CL$LOAD_SECONDARY NOVALUE =
801      0795 1 ++
802      0796 1
803      0797 1 Functional Description:
804      0798 1
805      0799 1 Loads a secondary key of an index sequential file. Which secondary
806      0800 1 index depends on KEY_REF. The secondary
807      0801 1 data records are read from the RFA file created and opened by
808      0802 1 SORT_SECONDARY. NOTE: The overall operation of LOAD_SECONDARY is
809      0803 1 fundamentally different than LOAD_PRIMARY.
810      0804 1
811      0805 1 Calling Sequence:
812      0806 1
813      0807 1     CONV$LOAD_SECONDARY();
814      0808 1
815      0809 1 Input Parameters:
816      0810 1     none
817      0811 1
818      0812 1 Implicit Inputs:
819      0813 1     none
820      0814 1
821      0815 1 Output Parameters:
822      0816 1     none
823      0817 1
824      0818 1 Implicit Outputs:
825      0819 1     none
826      0820 1
827      0821 1 Routine Value:
828      0822 1
829      0823 1     RMSS_EOF or error codes
830      0824 1
831      0825 1 Routines Called:
832      0826 1
833      0827 1     CONV$GET_BUCKET
834      0828 1     CONV$GET_TEMP_VM
835      0829 1     $GET
836      0830 1     CONV$CHECK_NULL
837      0831 1     CONV$CHECK_S_DUP
838      0832 1     LOAD DATA BUCKET
839      0833 1     CONV$COPY_KEY
840      0834 1     CONV$WRITE_BUCKET
841      0835 1     CONV$INIT_BUCKET
842      0836 1     CONV$CONVERT_VBN_ID
843      0837 1     FINISH_INDEX
844      0838 1
845      0839 1 Side Effects:
846      0840 1
847      0841 1     Loads secondary index defined by KEY_REF
848      0842 1
849      0843 1     --
850      0844 1
851      0845 2 BEGIN
852      0846 2
853      0847 2     DEFINE_CTX;
854      0848 2     DEFINE_BUCKET;
855      0849 2     DEFINE_KEY_DESC;
```

```
856      0850 2
857      0851 2      LABEL
858      0852 2      NULL_BLK;
859      0853 2
860      0854 2
861      0855 2      LOCAL
862      0856 2      DUP_COUNT,
863      0857 2      MAX_NUM_DUP;
864      0858 2      | Init some values
865      0859 2
866      0860 2      CONTINUATION = _CLEAR;
867      0861 2      DUPLICATE = _CLEAR;
868      0862 2
869      0863 2      | Errors on the rab from now on are WRITEERRS
870      0864 2
871      0865 2      CONV$AB_OUT_RAB [ RABSL_CTX ] = CONV$_WRITEERR;
872      0866 2
873      0867 2      | Point to the first block
874      0868 2
875      0869 2      CTX = .CONV$GL_CTX_BLOCK;
876      0870 2
877      0871 2      | Get the Buckets for the secondary data area and at least the
878      0872 2      first level of the index
879      0873 2
880      0874 2      Get the bucket for level 0
881      0875 2
882      0876 2      CONV$$GET_BUCKET( .KEY_DESC [ KEY$B_DANUM ] );
883      0877 2
884      0878 2      KEY_DESC [ KEY$L_LDVBIN ] = .CTX [ CTX$L_CURRENT_VBN ];
885      0879 2
886      0880 2      | Get the bucket for level 1
887      0881 2
888      0882 2      CTX = .CTX + CTX$K_BLN;
889      0883 2      CONV$$GET_BUCKET( .KEY_DESC [ KEY$B_LANUM ] );
890      0884 2
891      0885 2      CTX = .CONV$GL_CTX_BLOCK;
892      0886 2
893      0887 2      | Before we start we need to calculate the size of the level 0 index record
894      0888 2      buffer. This calculation is VERY important it must be very accurate!
895      0889 2
896      0890 2      If we allow dup. keys the it becomes complicated
897      0891 2
898      0892 2      Find out the max. number of duplicates that can fit in this bucket
899      0893 2
900      0894 2      IF .KEY_DESC [ KEY$V_DUPKEYS ]
901      0895 2      THEN
902      0896 2
903      0897 2      | Sizes are different for prologue 3
904      0898 2
905      0899 2      IF .CONV$GB_PROL_V3
906      0900 2      THEN
907      0901 2
908      0902 2      | For compression it is also different
909      0903 2
910      0904 2      IF .KEY_DESC [ KEY$V_IDX_COMPR ]
911      0905 2      THEN
912      0906 2
```

```

913    0907 2      | The space in the bucket minus the key size and the record
914    0908 2      | overhead (2+2) divided by the size of the SIDR record
915    0909 2
916    0910 2
917    0911 3      MAX_NUM_DUP = ( .CTX [ CTX$W_SPC ] -
918    0912 2          ( .KEY_DESC [ KEY$B_KEYSZ ] + 4 ) ) / 7
919    0913 2      ELSE
920    0914 2
921    0915 2      | The space in the bucket minus the key size and the record
922    0916 2      | overhead (2) divided by the size of the SIDR record
923    0917 2      | pointer (7)
924    0918 2
925    0919 3      MAX_NUM_DUP = ( .CTX [ CTX$W_SPC ] -
926    0920 2          ( .KEY_DESC [ KEY$B_KEYSZ ] + 2 ) ) / 7
927    0921 2      ELSE
928    0922 2
929    0923 2      | The space in the bucket minus the key size and the record
930    0924 2      | overhead (8) divided by the size of the SIDR record
931    0925 2      | pointer (6)
932    0926 2
933    0927 2      MAX_NUM_DUP = ( .CTX [ CTX$W_SPC ] -
934    0928 3          ( .KEY_DESC [ KEY$B_KEYSZ ] + 8 ) ) / 6
935    0929 2      ELSE
936    0930 2          MAX_NUM_DUP = 1;
937    0931 2
938    0932 2
939    0933 3      BEGIN
940    0934 2
941    0935 2      LOCAL      BYTES;
942    0936 2
943    0937 3      | The size of the level 0 buffer consist of:
944    0938 2
945    0939 2      | Space for all RRVs (one for each dup) : Largest rrv - prologue 3, 7 bytes
946    0940 2      | Overhead : Maximun overhead - prologue 1, 8 bytes
947    0941 2
948    0942 2      | We also need a duplicate buffer for things which is the size of the key
949    0943 2
950    0944 2      BYTES = ( .MAX_NUM_DUP * 7 ) + 8 + .KEY_DESC [ KEY$B_KEYSZ ];
951    0945 2
952    0946 2      | Allocate the memory for the buffer
953    0947 2      | The level 0 data record pointers points to this buffer
954    0948 2
955    0949 2      CTX [ CTX$L_RDP ] = CONV$$GET_TEMP_VM ( .BYTES );
956    0950 2
957    0951 2      | The duplicate buffer is just past that
958    0952 2
959    0953 2      CONV$GL_DUP_BUF = .CTX [ CTX$L_RDP ] + ( .MAX_NUM_DUP * 7 ) + 8
960    0954 2
961    0955 2      END;
962    0956 2
963    0957 2      | For the secondary key the Data comes from $GET on the RFA RAB
964    0958 2
965    0959 2      BEGIN          ! RECORD_CTRL local
966    0960 2
967    0961 2      DEFINE_RECORD_CTRL_GLOBAL;
968    0962 2
969    0963 3      LOCAL

```

```
: 970      0964 3      ALL_NULL,  
: 971      0965           SKIP,  
: 972      0966           STATUS;  
: 973      0967  
: 974      0968           SKIP = _CLEAR;  
: 975      0969  
: 976      0970           RECORD_CTRL = .CTX [ CTXSL_RCP ];  
: 977      0971  
: 978      0972           ALL_NULL = _SET;    ! Could be nothing but null keys, you know...  
: 979      0973  
: 980      0974           ! Main record processing loop. The size of the record is returned in  
: 981      0975           ! RFA_RAB [ RAB$W_RSZ ]  
: 982      0976  
: 983      0977 4      WHILE ( STATUS = $GET( RAB=CONV$AB_RFA_RAB ) )  
: 984      0978 3      DO  
: 985      0979 4      BEGIN          ! Main Loop  
: 986      0980 4  
: 987      0981 4      NULL_BLK:  
: 988      0982 5      BEGIN          ! NULL_BLK null key value block  
: 989      0983 5  
: 990      0984 5      LOCAL DUP;  
: 991      0985 5  
: 992      0986 5      ! If the record is too short (does not contain the complete key)  
: 993      0987 5      then treat it as a null key  
: 994      0988 5  
: 995      0989 5      IF ( .CONV$AB_RFA_RAB [ RAB$W_RSZ ] - 6 ) LSSU .KEY_DESC [ KEY$B_KEYSZ ]  
: 996      0990 5      THEN  
: 997      0991 5      LEAVE NULL_BLK;  
: 998      0992 5  
: 999      0993 5      ! If the file allows null keys check to see if this is one  
: 1000     0994 5  
: 1001     0995 5      IF .KEY_DESC [ KEY$V_NULKEYS ]  
: 1002     0996 5      THEN  
: 1003     0997 5  
: 1004     0998 5      ! If this is a null key then just ignore this record  
: 1005     0999 5  
: 1006     1000 5      IF CONV$$CHECK_NULL()  
: 1007     1001 5      THEN  
: 1008     1002 5      LEAVE NULL_BLK;  
: 1009     1003 5  
: 1010     1004 5  
: 1011     1005 5      ! If we got a non-null key, then all_null can no longer be true  
: 1012     1006 5  
: 1013     1007 5      IF .ALL_NULL THEN ALL_NULL = _CLEAR;  
: 1014     1008 5  
: 1015     1009 5      ! Check to see if this is a duplicate.  
: 1016     1010 5  
: 1017     1011 5      DUP = CONV$$CHECK_S_DUP();  
: 1018     1012 5  
: 1019     1013 5      ! Process the key  
: 1020     1014 5  
: 1021     1015 5      IF .KEY_DESC [ KEY$V_DUPKEYS ]  
: 1022     1016 5      THEN  
: 1023     1017 6      BEGIN  
: 1024     1018 6  
: 1025     1019 6      ! If this was a dup  
: 1026     1020 6
```

```
: 1027      1021 6      IF .DUP
: 1028      1022 6      THEN BEGIN
: 1029      1023 7
: 1030      1024 7
: 1031      1025 7      DUP_COUNT = .DUP_COUNT + 1;
: 1032      1026 7
: 1033      1027 7      | If we have exceeded the max number of dups per bucket then
: 1034      1028 7      | get rid of this bucket and start a new one
: 1035      1029 7
: 1036      1030 7      IF .DUP_COUNT GEQ .MAX_NUM_DUP
: 1037      1031 7      THEN BEGIN
: 1038      1032 8
: 1039      1033 8
: 1040      1034 8      LOAD_DATA_BUCKET();
: 1041      1035 8
: 1042      1036 8      | The record to go into the next bucket will be a duplicate
: 1043      1037 8
: 1044      1038 8      DUPLICATE = _SET;
: 1045      1039 8
: 1046      1040 8      | We are now in a continuation bucket
: 1047      1041 8
: 1048      1042 8      SKIP = _SET;
: 1049      1043 8
: 1050      1044 8      | Copy the key into the record (in a continuation bucket
: 1051      1045 8      | there is no dup count ie. the 4)
: 1052      1046 8
: 1053      1047 8      CONV$COPY_KEY( 4 );
: 1054      1048 8
: 1055      1049 8      | Start counting dups again
: 1056      1050 8
: 1057      1051 8      DUP_COUNT = 0;
: 1058      1052 8
: 1059      1053 8      | Set the sidr array record size
: 1060      1054 8
: 1061      1055 8      CTX [ CTX$W_RDS ] = 0;
: 1062      1056 8
: 1063      1057 8      | Set some control fields. NOTE: COPY_KEY sets prologue 3
: 1064      1058 8      | record size field.
: 1065      1059 8
: 1066      1060 8      IF NOT .CONV$GB_PROL_V3
: 1067      1061 8      THEN BEGIN
: 1068      1062 9
: 1069      1063 9
: 1070      1064 9      | A continuation record has no duplicate pointer
: 1071      1065 9
: 1072      1066 9      RECORD_CTRL [ IRC$B_CONTROL ] = IRC$M_NOPTRSZ;
: 1073      1067 9
: 1074      1068 9      | Prologue 1,2 size field includes a key
: 1075      1069 9
: 1076      1070 9      RECORD_CTRL [ IRC$W_NODUPSZ ] = .KEY_DESC [ KEY$B_KEYSZ ]
: 1077      1071 9
: 1078      1072 9      END
: 1079      1073 8      END
: 1080      1074 7      ELSE BEGIN
: 1081      1075 6
: 1082      1076 7
: 1083      1077 7
```

```
: 1084    1078 7      ! If this is the first non-dup then don't load anything else
: 1085    1079 7      load the last record processed
: 1086    1080 7
: 1087    1081 7      IF NOT .CTX [ CTX$V_FST ]
: 1088    1082 7      THEN
: 1089    1083 8      BEGIN
: 1090    1084 8      LOAD_DATA_BUCKET();
: 1091    1085 8      ! The next record will not be a duplicate record
: 1092    1086 8      DUPLICATE = _CLEAR;
: 1093    1087 8      ! If we were in a continuatio bucket then dont make an index
: 1094    1088 8      for it. Also write the bucket because we don't put anything
: 1095    1089 8      in a bucket after a dup.
: 1096    1090 8
: 1097    1091 8
: 1098    1092 8
: 1099    1093 8
: 1100    1094 8
: 1101    1095 8      IF .SKIP
: 1102    1096 8      THEN
: 1103    1097 9      BEGIN
: 1104    1098 9      SKIP = _CLEAR;
: 1105    1099 9
: 1106    1100 9      CONV$SWRITE_BUCKET();
: 1107    1101 9      CONV$INIT_BUCKET();
: 1108    1102 9
: 1109    1103 9      ! The next record will always fit into the new bucket
: 1110    1104 9      so clearing the continuation flag is ok
: 1111    1105 9      CONTINUATION = _CLEAR
: 1112    1106 9
: 1113    1107 9
: 1114    1108 9
: 1115    1109 7      END
: 1116    1110 7      END;

: 1117    1111 7      ! Copy the key into the record past the dup count field (ie the 8)
: 1118    1112 7      CONV$COPY_KEY( 8 );
: 1119    1113 7
: 1120    1114 7      ! Start counting the dups
: 1121    1115 7      DUP_COUNT = 0;
: 1122    1116 7
: 1123    1117 7
: 1124    1118 7
: 1125    1119 7      ! Set the sidr array record size
: 1126    1120 7
: 1127    1121 7      CTX [ CTX$W_RDS ] = 0;
: 1128    1122 7
: 1129    1123 7      ! Set some control fields. NOTE: COPY_KEY sets prologue 3
: 1130    1124 7      record size field.
: 1131    1125 7
: 1132    1126 7      IF NOT .CONV$GB_PROL_V3
: 1133    1127 7      THEN
: 1134    1128 8      BEGIN
: 1135    1129 8      ! The size of the dup pointer (1=4bytes)
: 1136    1130 8      RECORD_CTRL [ IRC$B_CONTROL ] = 1;
: 1137    1131 8
: 1138    1132 8
: 1139    1133 8
: 1140    1134 8      ! Zero the field (not implemented)
```

```
1141      1135 8          !  
1142      1136 8          RECORD_CTRL [ IRC$L_DUPCOUNT ] = 0;  
1143      1137 8          ! Prologue 1,2 size field includes a key  
1144      1138 8          RECORD_CTRL [ IRC$W_DUPSZ ] = .KEY_DESC [ KEY$B_KEYSZ ]  
1145      1139 8          END  
1146      1140 8          END;  
1147      1141 8          ! Add to the size of the dup for this record.  
1148      1142 8          ( IF .CONV$GB_PROL_V3  
1149      1143 8          THEN  
1150      1144 6          ! A prologue 3 RRV is 7 bytes (1 control,2 ID,4 VBN)  
1151      1145 6          RECORD_CTRL [ IRC$W_P3SZ ] = .RECORD_CTRL [ IRC$W_P3SZ ] + 7  
1152      1146 6          ELSE  
1153      1147 7          BEGIN  
1154      1148 7          ! A prologue 1,2 RRV is 6 bytes (1 control,1 ID,4 VBN )  
1155      1149 7          IF .RECORD_CTRL [ IRC$V_NOPTRSZ ]  
1156      1150 7          THEN  
1157      1151 7          RECORD_CTRL [ IRC$W_NODUPSZ ] =  
1158      1152 7          .RECORD_CTRL [ IRC$W_NODUPSZ ] + 6  
1159      1153 7          ELSE  
1160      1154 7          RECORD_CTRL [ IRC$W_DUPSZ ] =  
1161      1155 7          .RECORD_CTRL [ IRC$W_DUPSZ ] + 6  
1162      1156 8          END )  
1163      1157 8          END  
1164      1158 8          ELSE  
1165      1159 8          BEGIN  
1166      1160 8          ! If the keys are duplicate and we are not allowing dup. then error  
1167      1161 8          IF .DUP  
1168      1162 8          THEN  
1169      1163 8          SIGNAL_STOP( CONV$LOADIDX,  
1170      1164 8          .KEY_DESC [ KEY$B_KEYREF ],  
1171      1165 8          RMSS_DUP );  
1172      1166 8          ! If this is the first record don't load anything else load the  
1173      1167 7          last record  
1174      1168 6          IF NOT .CTX [ CTX$V_FST ]  
1175      1169 6          THEN  
1176      1170 5          LOAD_DATA_BUCKET();  
1177      1171 6          ! Move the key value  
1178      1172 6          CONV$COPY_KEY( 4 );
```

```
1198      1192    6
1199      1193    6      ! Set the sidr array record size
1200      1194    6
1201      1195    6      CTX [ CTX$W_RDS ] = 0;
1202      1196    6
1203      1197    6      ! Set some control fields. NOTE: COPY_KEY sets prologue 3 record
1204      1198    6      size field NOT counting the pointer so we must add it here
1205      1199    6
1206      1200    6      IF .CONV$GB_PROL_V3
1207      1201    6      THEN
1208      1202    6      RECORD_CTRL [ IRCSW_P3SZ ] = .RECORD_CTRL [ IRCSW_P3SZ ] + 7
1209      1203    6      ELSE
1210      1204    7      BEGIN
1211      1205    7
1212      1206    7      ! Non dup records don't have a dup count
1213      1207    7
1214      1208    7      RECORD_CTRL [ IRCSB_CONTROL ] = IRCSM_NOPTRSZ;
1215      1209    7
1216      1210    7      RECORD_CTRL [ IRCSW_NODUPSZ ] = .KEY_DESC [ KEYSB_KEYSZ ] + 6
1217      1211    7
1218      1212    7      END
1219      1213    7
1220      1214    5
1221      1215    5
1222      1216    5      ! Load the SIDR array pointer
1223      1217    5
1224      1218    6      BEGIN ! SIDR local
1225      1219    6
1226      1220    6      DEFINE_VBN_ID_GLOBAL;
1227      1221    6
1228      1222    6      LOCAL SIDR : REF BLOCK [ ,BYTE ];
1229      1223    6
1230      1224    6      ! Convert the VBN and the ID that SORT returns in the file
1231      1225    6
1232      1226    6      CONV$CONVERT_VBN_ID();
1233      1227    6
1234      1228    6      ! Move the record pointer right after the last one, if any
1235      1229    6
1236      1230    6      SIDR = .CTX [ CTX$L_RDP ] + .CTX [ CTX$W_RDS ];
1237      1231    6
1238      1232    6      ! If prologue 3 the ID is bigger
1239      1233    6
1240      1234    6      IF .CONV$GB_PROL_V3
1241      1235    6      THEN
1242      1236    7      BEGIN
1243      1237    7
1244      1238    7      ! Set the first_key flag if necessary
1245      1239    7
1246      1240    7      IF .DUP
1247      1241    7      THEN
1248      1242    7      SIDR [ 0,0,8,0 ] = 2          ! Can't be first if dup
1249      1243    7      ELSE
1250      1244    7      SIDR [ 0,0,8,0 ] = 2 + IRCSM_FIRST_KEY; ! Set flag and size
1251      1245    7
1252      1246    7      SIDR [ 1,0,16,0 ] = .SORT_ID;
1253      1247    7      SIDR [ 3,0,32,0 ] = :SORT_VBN;
1254      1248    7      CTX [ CTX$W_RDS ] = .CTX [ CTX$W_RDS ] + 7
```

```
; 1255      1249 7      ELSE END
; 1256      1250 6
; 1257      1251 7      BEGIN
; 1258      1252 7      SIDR [ 0,0,8,0 ] = 2;
; 1259      1253 7      SIDR [ 1,0,8,0 ] = :SORT_ID;
; 1260      1254 7      SIDR [ 2,0,32,0 ] = :SORT_VBN;
; 1261      1255 7      CTX [ CTXSW_RDS ] = :CTX [ CTXSW_RDS ] + 6
; 1262      1256 7      END
; 1263      1257 7
; 1264      1258 5      END;           ! SIDR local
; 1265      1259 5
; 1266      1260 5      | If we are here then we have processed at least one non null record
; 1267      1261 5
; 1268      1262 5      CTX [ CTX$V_FST ] = _CLEAR;
; 1269      1263 5
; 1270      1264 5      | If this is a non dup key then copy the current record into
; 1271      1265 5      | dup buffer
; 1272      1266 5
; 1273      1267 5      IF NOT .DUP
; 1274      1268 5      THEN
; 1275      1269 5      CH$MOVE( .KEY_DESC [ KEY$B KEYSZ ],
; 1276      1270 5      :CONV$GL_RFA_BUFFER + 6,
; 1277      1271 5      :CONV$GL_DUP_BUF )
; 1278      1272 5
; 1279      1273 5      END           ! NULL_BLK null key value block
; 1280      1274 5
; 1281      1275 3      END;           ! Main loop
; 1282      1276 3
; 1283      1277 3
; 1284      1278 3      | If we exited because of end of file AND we got at least 1
; 1285      1279 3      | non-null key value, then finish off the index
; 1286      1280 3      IF .STATUS EQL RMSS_EOF AND NOT .ALL_NULL
; 1287      1281 3      THEN
; 1288      1282 4      BEGIN
; 1289      1283 4
; 1290      1284 4      | There is a SIDR record left over at this point
; 1291      1285 4      | We must load it in before we finish off the index
; 1292      1286 4
; 1293      1287 4      LOAD_DATA_BUCKET();
; 1294      1288 4
; 1295      1289 4      FINISH_INDEX()
; 1296      1290 4
; 1297      1291 3
; 1298      1292 3
; 1299      1293 3
; 1300      1294 3
; 1301      1295 3      END           ! RECORD_CTRL local
; 1302      1296 3
; 1303      1297 1      END:
; INFO#250          L1:1025
; Referenced LOCAL symbol DUP_COUNT is probably not initialized
```

.EXTRN SY\$GET

01FC 8F BB 00000 CONV\$LOAD\_SECONDARY::

				PUSHR #^M<R2,R3,R4,R5,R6,R7,R8>	0794
				SUBL2 #20 SP	0860
				CLRW CONTINUATION	0865
				MOVL #CONVS WRITEERR, CONVSAB_OUT_RAB+24	0869
				MOVBL CONV\$GL CTX_BLOCK CTX	0876
				MOVZBL 8(KEY_DESC), -(SP)	0878
				CONV\$GET_BUCKET	0882
				8(CTX), 84(KEY_DESC)	0883
				MOVAB 92(R10), CTX	0885
				MOVZBL 7(KEY DESC), (SP)	0889
				CONV\$GET_BUCKET	0894
				ADDL2 #4, SP	0904
				CONV\$GL CTX_BLOCK, CTX	0912
				MOVBL 16(KEY DESC), 4\$	0911
				BLBC CONV\$GB PROL V3, 3\$	0912
				#3, 16(KEY DESC), 1\$	0912
				42(CTX), R0	0920
				MOVZWL 20(KEY DESC), R1	0920
				SUBL2 R1, R0	0920
				SUBL2 #4, R0	0920
				BRB 2\$	0920
				MOVZWL 42(CTX), R0	0920
				MOVZBL 20(KEY DESC), R1	0920
				SUBL2 R1, R0	0920
				SUBL2 #2, R0	0920
				DIVL3 #7, R0, MAX_NUM_DUP	0920
				BRB 5\$	0920
				MOVZWL 42(CTX), R0	0929
				MOVZBL 20(KEY DESC), R1	0929
				SUBL2 R1, R0	0929
				SUBL2 #8, R0	0929
				DIVL3 #6, R0, MAX_NUM_DUP	0929
				BRB 5\$	0929
				MOVBL #1, MAX_NUM_DUP	0931
				MULL3 #7, MAX_NUM_DUP, R1	0944
				MOVZBL 20(KEY DESC), R0	0944
				MOVAB 8(R0)[R1], BYTES	0949
				PUSHL BYTES	0949
				BSBW CONV\$GET_TEMP_VM	0949
				ADDL2 #4, SP	0953
				MOVBL R0, 52(CTX)	0953
				ADDL3 52(CTX), R1, R0	0968
				MOVAB 8(R0), CONV\$GL_DUP_BUF	0970
				CLRL SKIP	0972
				MOVL 48(CTX), RECORD_CTRL	0977
				MOVL #1, ALL_NULL	0977
				PUSHAB CONVSAB_RFA_RAB	0977
				CALLS #1, SYSSGET	0977
				MOVL R0, STATUS	0977
				BLBS STATUS, 7\$	0977
				BRW 25\$	0977
				MOVZWL CONVSAB_RFA_RAB+34, R0	0989
				SUBL2 #6, R0	0989
				CMPZV #0, #8, 20(KEY DESC), R0	0989
				BGTRU 6\$	0995
				BBC #2, 16(KEY DESC), 8\$	1000
				CONV\$CHECK_NULL	1000

CONVSFSTLD  
V04-000

VAX-11 CONVERT  
LOAD\_SECONDARY

I 13  
15-Sep-1984 23:49:35 VAX-11 Bliss-32 v4.0-742  
14-Sep-1984 12:14:00 [CONV.SRC]CONVFSTLD.B32;1

Page 32  
(7)

CONVSFSTLD  
V04-000

VAX-11 CONVERT  
LOAD SECONDARY

J 13  
15-Sep-1984 23:49:35  
14-Sep-1984 12:14:00

VAX-11 Bliss-32 V4.0-742  
[CONV.SRC]CONVFSTLD.B32;1

Page 33  
(7)

02	68		OC	11	001A4		BRB	19\$			1208
02	A8	14	AB	9B	001A6	17\$:	MOV B	#16, (RECORD_CTRL)			1210
02	A8		06	A0	001AE	18\$:	MOVZBW	20(KEY DESC), 2(RECORD_CTRL)			1226
		0000G	30	001B2		19\$:	ADDW2	#6, 2(RECORD_CTRL)			1230
	50	3A	AA	3C	001B5		BSBW	CONV\$CONVERT_VBN_ID			1234
	50	34	AA	C0	001B9		MOVZWL	58(CTX), SIDR			1240
	1A	0000G	CF	E9	001BD		BLBC	52(CTX), SIDR			1242
	05		52	E9	001C2		BLBC	CONV\$GB_PROL_V3, 22\$			1244
	60		02	90	001C5		MOVB	DUP, 20\$			1246
			04	11	001C8		BRB	#2, (SIDR)			1248
01	60	82	8F	90	001CA	20\$:	MOV B	21\$, #126, (SIDR)			1244
03	A0		57	B0	001CE	21\$:	MOVW	SORT_ID, 1(SIDR)			1246
3A	AA		56	D0	001D2		MOVL	SORT_VBN, 3(SIDR)			1247
			07	A0	001D6		ADDW2	#7, 58(CTX)			1248
			0F	11	001DA		BRB	23\$			
01	60		02	90	001DC	22\$:	MOV B	#2, (SIDR)			1252
02	A0		57	90	001DF		MOV B	SORT_ID, 1(SIDR)			1253
3A	AA		56	D0	001E3		MOVL	SORT_VBN, 2(SIDR)			1254
			06	A0	001E7		ADDW2	#6, 58(CTX)			1255
	6A		01	8A	001EB	23\$:	BICB2	#1, (CTX)			1262
	10		52	E8	001EE		BLBS	DUP, 24\$			1267
	51	14	AB	9A	001F1		MOVZBL	20(KEY DESC), R1			1269
0000' DF	06	A0	0000G	CF	D0	001F5	MOVL	CONV\$GL_RFA_BUFFER, R0			1270
			51	28	001FA		MOVC3	R1, 6(R0), &CONV\$GL_DUP_BUF			1271
0001827A	8F	10	FEAC	31	00201	24\$:	BRW	6\$			0979
			AE	D1	00204	25\$:	CMPL	STATUS, #98938			1280
	06	08	0A	12	0020C		BNEQ	26\$			
			AE	E8	0020E		BLBS	ALL NULL, 26\$			1287
		0000V	30	00212		BSBW	LOAD DATA BUCKET				1289
		0000V	30	00215		BSBW	FINISH INDEX				1297
	5E	14	C0	00218		26\$:	ADDL2	#20, SP			
	01FC	8F	BA	0021B			POPR	#^M<R2,R3,R4,R5,R6,R7,R8>			
			05	0021F			RSB				

; Routine Size: 544 bytes, Routine Base: \_CONV\$FAST\_S + 0238

```
: 1305      1298 1 %SBTTL 'LOAD DATA BUCKET'
: 1306      1299 1 ROUTINE LOAD_DATA_BUCKET : CL$JSB_REG_8 NOVALUE =
: 1307      1300 1 ++
: 1308      1301 1
: 1309      1302 1 Functional Description:
: 1310      1303 1
: 1311      1304 1 Loads a data bucket independent of key of reference in the
: 1312      1305 1 index. On a call to LOAD_DATA_BUCKET a record is loaded into a bucket
: 1313      1306 1 and return. If the record for some reason does not fit into the current
: 1314      1307 1 bucket an index is made for the bucket and the bucket is written to the
: 1315      1308 1 output file. The written bucket is initialized and then loaded with
: 1316      1309 1 the original record. The index for a bucket is made by calling
: 1317      1310 1 LOAD_INDEX_BUCKET.
: 1318      1311 1
: 1319      1312 1 Calling Sequence:
: 1320      1313 1
: 1321      1314 1     LOAD_DATA_BUCKET();
: 1322      1315 1
: 1323      1316 1 Input Parameters:
: 1324      1317 1     none
: 1325      1318 1
: 1326      1319 1 Implicit Inputs:
: 1327      1320 1
: 1328      1321 1 Output Parameters:
: 1329      1322 1     none
: 1330      1323 1
: 1331      1324 1 Implicit Outputs:
: 1332      1325 1     none
: 1333      1326 1
: 1334      1327 1 Routine Value:
: 1335      1328 1
: 1336      1329 1     SSSNORMAL or error codes
: 1337      1330 1
: 1338      1331 1 Routines Called:
: 1339      1332 1
: 1340      1333 1     CONV$$GET_BUCKET
: 1341      1334 1     LOAD_INDEX_BUCKET
: 1342      1335 1     CONV$$SAVE_BUCKET
: 1343      1336 1     CONV$$WRITE_BUCKET
: 1344      1337 1     CONV$$INIT_BUCKET
: 1345      1338 1     CONV$$RESTORE_BUCKET
: 1346      1339 1     CONV$$COMPRESS_KEY
: 1347      1340 1     CONV$$MAKE_INDEX
: 1348      1341 1     CONV$$WRITE_VBN
: 1349      1342 1
: 1350      1343 1 Side Effects:
: 1351      1344 1
: 1352      1345 1     Loads a record into a bucket. Writes buckets and creates indexes
: 1353      1346 1     for lower level buckets
: 1354      1347 1
: 1355      1348 1     --
: 1356      1349 1
: 1357      1350 2 BEGIN
: 1358      1351 2
: 1359      1352 2     DEFINE_CTX;
: 1360      1353 2     DEFINE_BUCKET;
: 1361      1354 2     DEFINE_KEY_DESC;
```

```

1362      1355 2   DEFINE_RECORD_CTRL;
1363      1356 2
1364      1357 2   | Set the bucket pointer to the bucket at this level
1365      1358 2
1366      1359 2   BUCKET = .CTX [ CTX$L_CURRENT_BUFFER ];
1367      1360 2
1368      1361 2   | Will the record fit into the bucket, if not then call this thing
1369      1362 2   with an index to the record.
1370      1363 2
1371      1364 2   | A record will not fit into a bucket if:
1372      1365 2
1373      1366 2   |   For all files:
1374      1367 2
1375      1368 2   |   a) the combined record data size and record control size is greater than
1376      1369 2   |       the space available in the bucket.
1377      1370 2
1378      1371 2   |   b) the FILL switch is OFF and the space left in the bucket is less than
1379      1372 2   |       that allowed by bucket fill quantities
1380      1373 2
1381      1374 2   |   For prologue 1 & 2 files:
1382      1375 2
1383      1376 2   |   c) the record ID of the new record is 0 indicating that the bucket is
1384      1377 2   |       filled (as far as id are concerned)
1385      1378 2
1386      1379 4   IF ( ( ( .CTX [ CTX$W_RDS ] + .CTX [ CTX$W_RCS ] ) GTRU
1387      1380 4           .CTX [ CTX$W_SPC ] )
1388      1381 3   OR
1389      1382 3
1390      1383 4   ( IF .CONV$GB_PROL_V3
1391      1384 4       THEN 0
1392      1385 4       ELSE .BUCKET [ BKT$B_NXTRECID ] EQLU 0 )
1393      1386 4
1394      1387 3   OR
1395      1388 3
1396      1389 4   ( ( NOT .CONV$GL_FILL ) AND
1397      1390 5   ( LOCAL
1398      1391 5       SPACE USED IF RECORD ADDED;
1399      1392 5       SPACE_USED_IF_RECORD_ADDED = .CTX[CTX$W_USE] + .CTX[CTX$W_RCS]
1400      1393 5           + .CTX[CTX$W_RDS];
1401      1394 5       IF .KEY_DESC[KEY$W_DATFILL] - .CTX[CTX$W_USE]
1402      1395 5           LEQ
1403      1396 5           .SPACE_USED_IF_RECORD_ADDED - .KEY_DESC[KEY$W_DATFILL]
1404      1397 5           THEN
1405      1398 5           TRUE
1406      1399 5           ELSE
1407      1400 5           FALSE
1408      1401 5       ) ) )
1409      1402 2
1410      1403 2
1411      1404 3   THEN
1412      1405 3       BEGIN ! Load index block
1413      1406 3
1414      1407 3   | If for some reason we dont want to make an index entry for this
1415      1408 3   | record then skip it.
1416      1409 3
1417      1410 3   IF NOT .CONTINUATION
1418      1411 4       THEN
1419      1420 4           BEGIN

```

! If the difference now  
(must be signed)  
is less than it would  
be if the record were added,  
then don't add it  
else go ahead and add it

```
; 1419    1412 4
; 1420    1413 4      | Increase the level number for the next index level
; 1421    1414 4      CTX = .CTX + CTX$K_BLN;
; 1422    1415 4
; 1423    1416 4
; 1424    1417 4      | Call to LOAD_INDEX_BUCKET to load the next level of the index
; 1425    1418 4
; 1426    1419 4      LOAD_INDEX_BUCKET();
; 1427    1420 4
; 1428    1421 4      | Return the level
; 1429    1422 4
; 1430    1423 4      CTX = .CTX - CTX$K_BLN;
; 1431    1424 4
; 1432    1425 4      | Restore the bucket pointer to the current level bucket since
; 1433    1426 4      we should be looking at some other one.
; 1434    1427 4
; 1435    1428 4      BUCKET = .CTX [ CTX$L_CURRENT_BUFFER ]
; 1436    1429 4
; 1437    1430 3      END:
; 1438    1431 3
; 1439    1432 3      ! Write the bucket we filled
; 1440    1433 3
; 1441    1434 3      CONV$$WRITE_BUCKET();
; 1442    1435 3
; 1443    1436 3      ! If this is a dup then the next bucket is a continuation bucket
; 1444    1437 3
; 1445    1438 3      IF .DUPLICATE
; 1446    1439 3      THEN CONTINUATION = _SET
; 1447    1440 3      ELSE CONTINUATION = _CLEAR;
; 1448    1441 3
; 1449    1442 3
; 1450    1443 3
; 1451    1444 3      ! Initialize the bucket to use it again
; 1452    1445 3
; 1453    1446 3      CONV$$INIT_BUCKET()
; 1454    1447 3
; 1455    1448 2      END;   ! Load index block
; 1456    1449 2
; 1457    1450 2      BEGIN      ! BKT_*_PTR local
; 1458    1451 2
; 1459    1452 2      ! Load the record into the bucket...
; 1460    1453 2      ! First we must set up pointers to where the record will go in the bucket
; 1461    1454 2      These are:
; 1462    1455 2
; 1463    1456 2
; 1464    1457 2      LOCAL
; 1465    1458 2      BKT_CTRL_PTR,  ! Control information
; 1466    1459 2      BKT_DATA_PTR;  ! Actual data record
; 1467    1460 2      ! For Prologue 3 files...
; 1468    1461 2
; 1469    1462 2      IF .CONV$GB_PROL_V3
; 1470    1463 2      THEN
; 1471    1464 2      BEGIN
; 1472    1465 2
; 1473    1466 2      ! If key compression is on do it
; 1474    1467 2
; 1475    1468 4      IF .KEY_DESC [ KEY$V_KEY_COMPR ]
```

```
1476      1469 4      THEN  
1477      1470 4      CONV$$COMPRESS_KEY();  
1478      1471 4      ! Key of ref. specific things  
1479      1472 4      IF .KEY_DESC [ KEY$B_KEYREF ] EQL 0  
1480      1473 4      THEN  
1481      1474 4      ! The Primary key...  
1482      1475 4      BEGIN  
1483      1476 4      ! The record ID  
1484      1477 4      RECORD_CTRL [ IRC$W_ID ] = .BUCKET [ BKT$W_NXTRECID ];  
1485      1478 4      ! The RRV points to it's self ie. it's own ID and VBN  
1486      1479 5      RECORD_CTRL [ IRC$W_RRV_ID ] = .BUCKET [ BKT$W_NXTRECID ];  
1487      1480 5      RECORD_CTRL [ IRC$L_RRV_VBN3 ] = .CTX [ CTX$L_CURRENT_VBN ];  
1488      1481 5      ! Update the record next record id in the bucket  
1489      1482 5      BUCKET [ BKT$W_NXTRECID ] = .BUCKET [ BKT$W_NXTRECID ] + 1  
1490      1483 5      END  
1491      1484 5      ELSE  
1492      1485 3      ! For prologue 1 and 2 files...  
1493      1486 3      BEGIN  
1494      1487 3      ! The record ID  
1495      1488 4      RECORD_CTRL [ IRC$B_ID ] = .BUCKET [ BKT$B_NXTRECID ];  
1496      1489 4      ! If this is the primary data level the set up the RRV  
1497      1490 4      IF .KEY_DESC [ KEY$B_KEYREF ] EQL 0  
1498      1491 4      THEN  
1499      1492 5      BEGIN  
1500      1493 5      ! The RRV points to itself ie. it's own ID and VBN  
1501      1494 5      RECORD_CTRL [ IRC$B_RRV_ID ] = .BUCKET [ BKT$B_NXTRECID ];  
1502      1495 5      RECORD_CTRL [ IRC$L_RRV_VBN ] = .CTX [ CTX$L_CURRENT_VBN ];  
1503      1496 3      END;  
1504      1497 3      ! Update the next record id in the bucket  
1505      1498 4      BUCKET [ BKT$B_NXTRECID ] = .BUCKET [ BKT$B_NXTRECID ] + 1  
1506      1499 4      END;  
1507      1500 3      ! For all data levels the control bytes are put at the bucket
```

```

: 1533      1526 3   ! freespace. The data bytes are put directly after the control.
: 1534      1527
: 1535      1528  BKT_CTRL_PTR = .BUCKET [ BKT$W_FREESPACE ] + .BUCKET;
: 1536      1529  BKT_DATA_PTR = .BKT_CTRL_PTR + .CTX [ CTX$W_RCS ];
: 1537      1530
: 1538      1531  ! Update the bucket pointer (NOTE: Same update for all cases)
: 1539      1532
: 1540      1533  BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] +
: 1541      1534          .CTX [ CTX$W_RCS ] +
: 1542      1535          .CTX [ CTX$W_RDS ];
: 1543      1536
: 1544      1537  ! Load the record into the bucket...
: 1545      1538  ! Move the control bytes into the bucket
: 1546      1539
: 1547      1540  CH$MOVE( .CTX [ CTX$W_RCS ],.CTX [ CTX$L_RCP ],.BKT_CTRL_PTR );
: 1548      1541
: 1549      1542  ! Move the data bytes (or sidr array) into the bucket
: 1550      1543
: 1551      1544  CH$MOVE( .CTX [ CTX$W_RDS ],.CTX [ CTX$L_RDP ],.BKT_DATA_PTR );
: 1552      1545
: 1553      1546 2   END;           ! BKT_*_PTR local
: 1554      1547 2
: 1555      1548 2   ! Update the amount of space left in the bucket and the amount used
: 1556      1549 2
: 1557      1550 3   BEGIN
: 1558      1551
: 1559      1552  LOCAL
: 1560      1553 3   SPACE_USED;
: 1561      1554
: 1562      1555 3   SPACE_USED = .CTX [ CTX$W_RCS ] + .CTX [ CTX$W_RDS ];
: 1563      1556
: 1564      1557 3   CTX [ CTX$W_SPC ] = .CTX [ CTX$W_SPC ] - .SPACE_USED;
: 1565      1558
: 1566      1559 3   CTX [ CTX$W_USE ] = .CTX [ CTX$W_USE ] + .SPACE_USED;
: 1567      1560
: 1568      1561 2
: 1569      1562 2
: 1570      1563 2   ! Make an index for the next level
: 1571      1564 2
: 1572      1565 2   CONV$$MAKE_INDEX();
: 1573      1566 2
: 1574      1567 2   ! Set the index record control bytes and bucket pointer
: 1575      1568 2
: 1576      1569 2   CONV$$WRITE_VBN();
: 1577      1570 2
: 1578      1571 2   RETURN
: 1579      1572 2
: 1580      1573 1   END;

```

	007C	8F	BB 00000 LOAD_DATA_BUCKET:	
59	04	AA	DO 00004	POSHR #^M<R2,R3,R4,R5,R6>
50	3A	AA	3C 00008	MOVL 4(CTX), BUCKET
				MOVZWL 58(CTX), R0

: 1299  
: 1359  
: 1379

			51	38	AA	3C 0000C	MOVZWL	56(CTX), R1	
			50	51	CO	00010	ADDL2	R1, R0	
			10	00	ED	00013	CMPZV	#0, #16, 42(CTX), R0	1380
			05	0000G	CF	E8 00019	BLSSU	2S	
			06	A9	95	00020	BLBS	CONV\$GB_PROL_V3, 1\$	1383
			2E	13	00023	TSTB	6(BUCKET)	1385	
			53	0000G	CF	E8 00025	BEQL	2S	
			50	2C	AA	3C 0002A	BLBS	CONV\$GL_FILL, 6\$	1389
			51	38	AA	3C 0002E	MOVZWL	44(CTX), R0	1392
			50	51	CO	00032	MOVZWL	56(CTX), R1	
			52	3A	AA	3C 00035	ADDL2	R1, R0	
			50	52	CO	00039	MOVZWL	58(CTX), R2	1393
			51	1A	AB	3C 0003C	ADDL2	R2, SPACE USED IF_RECORD_ADDED	
			52	2C	AA	3C 00040	MOVZWL	26(KEY DESC), R1	1394
			51	52	C2	00044	MOVZWL	44(CTX), R2	
			52	1A	AB	3C 00047	SUBL2	R2, R1	
			50	52	C2	0004B	MOVZWL	26(KEY_DESC), R2	1396
			50	51	D1	0004E	SUBL2	R2, R0	
				2A	14	00051	CMPL	R1, R0	
			0F	0000'	CF	E8 00053	BGTR	6\$	
			5A	5C	AA	9E 00058	BLBS	CONTINUATION, 3\$	1409
				0000V	30	0005C	MOVAB	92(R10), CTX	1415
			5A	A4	AA	9E 0005F	BSBW	LOAD INDEX BUCKET	1419
			59	04	AA	D0 00063	MOVAB	-92(R10), CTX	1423
				0000G	30	00067	MOVL	4(CTX), BUCKET	1428
			0000'	07	0000'	CF E9 0006A	BSBW	CONV\$SWRITE BUCKET	1434
					01	90 0006F	BLBC	DUPLICATE, 4\$	1438
					04	11 00074	MOVB	#1, CONTINUATION	1440
					0000'	CF 94 00076	BRB	5\$	
					0000G	30 0007A	CLRB	CONTINUATION	1442
			03	10	21	AB	BSBW	CONV\$INIT BUCKET	1446
					0000G	CF E9 0007D	BLBC	CONV\$GB_PROL_V3, 8\$	1462
					06	E1 00082	BBC	#6, 16(KEY DESC), 7\$	1468
					0000G	30 00087	BSBW	CONV\$\$COMPRESS_KEY	1470
					15	AB 95 0008A	TSTB	21(KEY_DESC)	1474
						2B 12 0008D	BNEQ	10\$	
				01	A8	06 A9 B0 0008F	MOVW	6(BUCKET), 1(RECORD_CTRL)	1483
				03	A8	06 A9 B0 00094	MOVW	6(BUCKET), 3(RECORD_CTRL)	1487
				05	A8	08 AA D0 00099	MOVL	8(CTX), 5(RECORD_CTRL)	1488
					06	A9 B6 0009E	INCW	6(BUCKET)	1492
						17 11 000A1	BRB	10\$	
				01	A8	06 A9 90 000A3	MOVB	6(BUCKET), 1(RECORD_CTRL)	1504
					15	AB 95 000A8	TSTB	21(KEY_DESC)	1508
				02	A8	06 A9 90 000AD	BNEQ	9\$	
				03	A8	08 AA D0 000B2	MOVB	6(BUCKET), 2(RECORD_CTRL)	1514
					06	A9 96 000B7	MOVL	8(CTX), 3(RECORD_CTRL)	1515
					51	04 A9 3C 000BA	INCBL	6(BUCKET)	1521
					51	59 CO 000BE	MOVZWL	4(BUCKET), BKT_CTRL_PTR	1528
					56	38 AA 3C 000C1	ADDL2	BUCKET, BKT_CTRL_PTR	
					56	51 CO 000C5	MOVZWL	56(CTX), BKT_DATA_PTR	1529
					50	04 A9 3C 000CB	ADDL2	BKT_CTRL_PTR_BKT_DATA_PTR	
					52	38 AA 3C 000CC	MOVZWL	4(BUCKET), R0	1534
				04	A9 50	52 CO 000D0	ADDL2	56(CTX), R2	
				61	30 BA	3A AA A1 000D3	ADDW3	R2, R0	
				66	34 BA	38 AA 28 000D9	MOVC3	58(CTX), R0, 4(BUCKET)	1535
						3A AA 28 000DF	MOVC3	56(CTX), @48(CTX), (BKT_CTRL_PTR)	1540
							MOVC3	58(CTX), @52(CTX), (BKT_DATA_PTR)	1544

CONV\$FSTLD  
V04-000

VAX-11 CONVERT  
LOAD\_DATA\_BUCKET

D 14

15-Sep-1984 23:49:35  
14-Sep-1984 12:14:00

VAX-11 Bliss-32 V4.0-742  
[CONV.SRC]CONVFSTLD.B32;1

Page 40  
(8)

50	38	AA	3C	000E5	MOVZWL	56(CTX), SPACE_USED	: 1555
51	3A	AA	3C	000E9	MOVZWL	58(CTX), R1	
50	51	CO	000ED		ADDL2	R1, SPACE USED	
2A AA	50	A2	000F0		SUBW2	SPACE_USED, 42(CTX)	1557
2C AA	50	A0	000F4		ADDW2	SPACE_USED, 44(CTX)	1559
	0000G	30	000F8		BSBW	CONV\$SMAKE INDEX	1565
	0000G	30	000FB		BSBW	CONV\$SWRITE VBN	1569
007C	8F	BA	000FE		POPR	#^M<R2,R3,R4,R5,R6>	
			05	00102	RSB		1573

; Routine Size: 259 bytes, Routine Base: \_CONV\$FAST\_S + 0458

```
: 1582      1574 1 %SBTTL 'LOAD_INDEX_BUCKET'
: 1583      1575 1 ROUTINE LOAD_INDEX_BUCKET : CL$JSB_REG_9 NOVALUE =
: 1584      1576 1 ++
: 1585      1577 1
: 1586      1578 1 Functional Description:
: 1587      1579 1
: 1588      1580 1 Loads an index bucket independent level in the index. On a
: 1589      1581 1 call to LOAD_INDEX_BUCKET a record is loaded into a bucket and
: 1590      1582 1 return. If the record for some reason does not fit into the current
: 1591      1583 1 bucket an index is made for the bucket and the bucket is written to the
: 1592      1584 1 output file. The written bucket is initialized and then loaded with
: 1593      1585 1 the original record. The index for a bucket is made by calling
: 1594      1586 1 LOAD_INDEX_BUCKET recursively. Each recursive call to LOAD_INDEX_BUCKET
: 1595      1587 1 is to moving up the index tree. CTX keeps track to where you are in
: 1596      1588 1 the tree. Most all variables are dependent on CTX so that the
: 1597      1589 1 context of each level is saved.
: 1598      1590 1
: 1599      1591 1 Calling Sequence:
: 1600      1592 1
: 1601      1593 1     LOAD_INDEX_BUCKET()
: 1602      1594 1
: 1603      1595 1 Input Parameters:
: 1604      1596 1     none
: 1605      1597 1
: 1606      1598 1 Implicit Inputs:
: 1607      1599 1     none
: 1608      1600 1
: 1609      1601 1 Output Parameters:
: 1610      1602 1     none
: 1611      1603 1
: 1612      1604 1 Implicit Outputs:
: 1613      1605 1     none
: 1614      1606 1
: 1615      1607 1 Routine Value:
: 1616      1608 1
: 1617      1609 1     SSSNORMAL or error codes
: 1618      1610 1
: 1619      1611 1 Routines Called:
: 1620      1612 1
: 1621      1613 1     CONV$$GET_BUCKET
: 1622      1614 1     LOAD_INDEX_BUCKET          - Recursive call
: 1623      1615 1     CONV$$WRITE_BUCKET
: 1624      1616 1     CONV$$INIT_BUCKET
: 1625      1617 1     CONV$$COMPRESS_INDEX
: 1626      1618 1     CONV$$WRITE_VBN
: 1627      1619 1
: 1628      1620 1 Side Effects:
: 1629      1621 1
: 1630      1622 1     Loads a record into a bucket. Writes buckets and creates indexes
: 1631      1623 1     for lower level buckets
: 1632      1624 1
: 1633      1625 1     --
: 1634      1626 1
: 1635      1627 2     BEGIN
: 1636      1628 2
: 1637      1629 2     DEFINE_CTX;
: 1638      1630 2     DEFINE_BUCKET;
```

```

1639      1631 2   DEFINE_KEY_DESC;
1640      1632 2
1641      1633 2   ! Set the bucket pointer to the bucket at this level
1642      1634 2
1643      1635 2   BUCKET = .CTX [ CTX$L_CURRENT_BUFFER ];
1644      1636 2
1645      1637 2   ! See if we have reached the maximum level. (If we have this is the
1646      1638 2   biggest file in the world!)
1647      1639 2
1648      1640 2   IF .CTX [ CTX$B_LEVEL ] GEQU MAX_IDX_LVL - 1
1649      1641 2   THEN
1650      1642 2       SIGNAL_STOP( CONV$_IDX_LIM );
1651      1643 2
1652      1644 2   Will the record fit into the bucket, if not then call this thing
1653      1645 2   with an index to the record.
1654      1646 2
1655      1647 2   A record will not fit into a bucket if:
1656      1648 2
1657      1649 2   For all files:
1658      1650 2
1659      1651 2   a) the combined record data size and record control size is greater than
1660      1652 2   the space available in the bucket.
1661      1653 2
1662      1654 2   b) the FILL switch is OFF and the space left in the bucket is less than
1663      1655 2   that allowed by bucket fill quantities
1664      1656 2
1665      1657 2   For prologue 3 files:
1666      1658 2
1667      1659 2
1668      1660 2   c) the bucket below has a different size vbn then this bucket (this
1669      1661 2   is to keep the same size vbn index buckets)
1670      1662 4   IF ( ( ( .CTX [ CTX$W_RDS ] + .CTX [ CTX$W_RCS ] ) GTRU
1671      1663 4           .CTX [ CTX$W_SPC ] )
1672      1664 3   OR
1673      1665 3
1674      1666 4   ( ( NOT .CONV$GL_FILL ) AND
1675      1667 5   ( LOCAL
1676      1668 5   SPACE_USED_IF_RECORD_ADDED;
1677      1669 5   SPACE_USED_IF_RECORD_ADDED = .CTX[CTX$W_USE] + .CTX[CTX$W_RCS]
1678      1670 5           + .CTX[CTX$W_RDS];
1679      1671 5   IF .KEY_DESC[KEY$W_IDXFILL] - .CTX[CTX$W_USE]
1680      1672 5           LEQ
1681      1673 5           .SPACE_USED_IF_RECORD_ADDED - .KEY_DESC[KEY$W_IDXFILL]      ! If the difference now
1682      1674 5           THEN
1683      1675 5           TRUE
1684      1676 5           ELSE
1685      1677 5           FALSE
1686      1678 4           )
1687      1679 4
1688      1680 3   OR
1689      1681 3
1690      1682 4   ( IF .CONV$GB_PROL_V3
1691      1683 4   THEN
1692      1684 5   ( LOCAL CTX_M1 : REF_BLOCK [ ,BYTE ];
1693      1685 5   CTX_M1 = .CTX - CTX$K_BLN;
1694      1686 5   IF .BUCKET [ BKT$V_PTR_SZ ] NEQU .CTX_M1 [ CTX$V_VBN ]
1695      1687 5   THEN 1

```

```
; 1696      1688      5      ELSE 0
; 1697      1689      5      )
; 1698      1690      3      ELSE 0 ) )
; 1699      1691      2      THEN
; 1700      1692      1      BEGIN ! Load index block
; 1701      1693      1      | Switch for the next index level
; 1702      1694      1      CTX = .CTX + CTX$K_BLN;
; 1703      1695      1      | See if the bucket in at the next level is ready if not get it ready
; 1704      1696      1      | IF NOT .CTX [ CTX$V_RDY ]
; 1705      1697      1      THEN
; 1706      1698      1      | Get the space for the bucket
; 1707      1699      1      CONV$$GET_BUCKET( .KEY_DESC [ KEY$B_IANUM ] );
; 1708      1700      1      | Recursive call to LOAD_INDEX_BUCKET to load the next level of the index
; 1709      1701      1      LOAD_INDEX_BUCKET();
; 1710      1702      1      | Return the level
; 1711      1703      1      CTX = .CTX - CTX$K_BLN;
; 1712      1704      1      | Restore the bucket pointer to the current level bucket since
; 1713      1705      1      we should be looking at some other one.
; 1714      1706      1      BUCKET = .CTX [ CTX$L_CURRENT_BUFFER ];
; 1715      1707      1      | Write the bucket we filled
; 1716      1708      1      CONV$$WRITE_BUCKET();
; 1717      1709      1      | Initialize the bucket to use it again
; 1718      1710      1      CONV$$INIT_BUCKET()
; 1719      1711      1      END; ! Load index block
; 1720      1712      1      BEGIN ! CTX_P1 local
; 1721      1713      1      LOCAL CTX_P1 : REF BLOCK [ ,BYTE ];
; 1722      1714      1      CTX_P1 = .CTX + CTX$K_BLN;
; 1723      1715      1      | An index record is made for levels 2 and above ( level 0 and 1 are
; 1724      1716      1      made by LOAD PRIMARY and LOAD SECONDARY depending on KEY_REF )
; 1725      1717      1      NOTE: Do this now because latter the key could get compressed
; 1726      1718      1      CH$MOVE( .CTX [ CTX$W_RDS ],..CTX [ CTX$L_RDP ],..CTX_P1 [ CTX$L_RDP ] );
; 1727      1719      1      | Set the size of the data record
; 1728      1720      1      CTX_P1 [ CTX$W_RDS ] = .CTX [ CTX$W_RDS ];
; 1729      1721      1
; 1730      1722      1
; 1731      1723      1
; 1732      1724      1
; 1733      1725      1
; 1734      1726      1
; 1735      1727      1
; 1736      1728      2
; 1737      1729      2
; 1738      1730      3
; 1739      1731      3
; 1740      1732      3
; 1741      1733      3
; 1742      1734      3
; 1743      1735      3
; 1744      1736      3
; 1745      1737      3
; 1746      1738      3
; 1747      1739      3
; 1748      1740      3
; 1749      1741      3
; 1750      1742      3
; 1751      1743      3
; 1752      1744      3
```

```
1753      1745      ! Set the size of the control record
1754      1746      IF .CONV$GB_PROL_V3
1755      1747      THEN
1756      1748      CTX_P1 [ CTX$W_RCS ] = .CTX [ CTX$V_VBN ] + 2
1757      1749      ELSE
1758      1750      CTX_P1 [ CTX$W_RCS ] = .CTX [ CTX$V_VBN ] + 3
1759      1751
1760      1752
1761      1753
1762      1754      END;          ! CTX_P1 local
1763      1755
1764      1756      BEGIN          ! BKT_*_PTR local
1765      1757
1766      1758      ! Load the record into the bucket...
1767      1759      First we must set up pointers to where the record will go in the bucket
1768      1760      These are:
1769      1761
1770      1762      LOCAL
1771      1763          BKT_CTRL_PTR,    ! Control information
1772      1764          BKT_DATA_PTR;   ! Actual data record
1773      1765
1774      1766      The reason we split them up is because prologue 3 files put the two pieces
1775      1767      in two different places depending on bucket type (ie. INDEX, PRIMARY data
1776      1768      and SECONDARY data bucket.
1777      1769
1778      1770      For Prologue 3 files...
1779      1771
1780      1772      IF .CONV$GB_PROL_V3
1781      1773      THEN
1782      1774          BEGIN
1783      1775
1784      1776          ! Prologue 3 files...
1785      1777
1786      1778          IF .KEY_DESC [ KEY$V_IDX_COMPR ]
1787      1779          THEN
1788      1780              CONV$COMPRESS_INDEX();
1789      1781
1790      1782          ! If level 1 save the pointers so we can backup latter
1791      1783
1792      1784          IF .BUCKET [ BKT$B_LEVEL ] EQLU 1
1793      1785          THEN
1794      1786              BEGIN
1795      1787                  SAVE_VBNFS = .BUCKET [ BKT$W_VBNFS ];
1796      1788                  SAVE_KEYFRESPC = .BUCKET [ BKT$W_KEYFRESPC ]
1797      1789              END;
1798      1790
1799      1791          ! Update this pointer first since we go backwards with it
1800      1792
1801      1793          BUCKET [ BKT$W_VBNFS ] = .BUCKET [ BKT$W_VBNFS ] - .CTX [ CTX$W_RCS ];
1802      1794
1803      1795          ! For the index levels the control bytes are put at the bucket
1804      1796          ! vbn freespace. The data bytes are put at the key free space.
1805      1797
1806      1798          BKT_CTRL_PTR = .BUCKET [ BKT$W_VBNFS ] + .BUCKET + 1;
1807      1799          BKT_DATA_PTR = .BUCKET [ BKT$W_KEYFRESPC ] + .BUCKET;
1808      1800
1809      1801          ! Update the rest of the bucket pointers
```

```
1810      1802    4      !  
1811      1803    4      BUCKET [ BKT$W_KEYFRESPC ] = .BUCKET [ BKT$W_KEYFRESPC ] +  
1812      1804    4          .CTX [ CTX$W_RDS ]  
1813      1805    4  
1814      1806    4      END  
1815      1807    3      ELSE  
1816      1808    3  
1817      1809    3      | For prologue 1 and 2 files...  
1818      1810    3  
1819      1811    4      BEGIN  
1820      1812    4  
1821      1813    4      | If level 1 save the pointers so we can backup latter  
1822      1814    4  
1823      1815    4      IF .BUCKET [ BKT$B_LEVEL ] EQLU 1  
1824      1816    4      THEN  
1825      1817    4          SAVE_FREESPACE = .BUCKET [ BKT$W_FREESPACE ];  
1826      1818    4  
1827      1819    4      | Set some pointers...  
1828      1820    4  
1829      1821    4      | For prologue 1 and 2 files the control bytes are put at the bucket  
1830      1822    4      freespace. The data bytes are put directly after the control.  
1831      1823    4  
1832      1824    4      BKT_CTRL_PTR = .BUCKET [ BKT$W_FREESPACE ] + .BUCKET;  
1833      1825    4      BKT_DATA_PTR = .BKT_CTRL_PTR + .CTX [ CTX$W_RCS ];  
1834      1826    4  
1835      1827    4      | Update the bucket pointer (NOTE: Same update for all cases)  
1836      1828    4  
1837      1829    4      BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] +  
1838      1830    4          .CTX [ CTX$W_RCS ] +  
1839      1831    4          .CTX [ CTX$W_RDS ];  
1840      1832    4  
1841      1833    3      END;  
1842      1834    3  
1843      1835    3  
1844      1836    3      | Load the record into the bucket...  
1845      1837    3      | Move the control bytes into the bucket  
1846      1838    3      CHSMOVE( .CTX [ CTX$W_RCS ],.CTX [ CTX$L_RCP ],.BKT_CTRL_PTR );  
1847      1839    3  
1848      1840    3  
1849      1841    3      | Move the data bytes into the bucket  
1850      1842    3      CHSMOVE( .CTX [ CTX$W_RDS ],.CTX [ CTX$L_RDP ],.BKT_DATA_PTR );  
1851      1843    3  
1852      1844    2      END;      ! BKT_*_PTR local  
1853      1845    2  
1854      1846    2  
1855      1847    2  
1856      1848    2  
1857      1849    2  
1858      1850    2  
1859      1851    3  
1860      1852    3  
1861      1853    3  
1862      1854    3  
1863      1855    3  
1864      1856    3  
1865      1857    3  
1866      1858    3      LOCAL  
1867          SPACE_USED;  
1868  
1869          SPACE_USED = .CTX [ CTX$W_RCS ] + .CTX [ CTX$W_RDS ];  
1870  
1871          CTX [ CTX$W_SPC ] = .CTX [ CTX$W_SPC ] - .SPACE_USED;  
1872  
1873          CTX [ CTX$W_USE ] = .CTX [ CTX$W_USE ] + .SPACE_USED;
```

```

: 1867    1859  2    END;
: 1868    1860  2
: 1869    1861  2    ! Set the index record control bytes and bucket pointer
: 1870    1862  2
: 1871    1863  2    CONV$SWRITE_VBN();
: 1872    1864  2
: 1873    1865  2    RETURN
: 1874    1866  2
: 1875    1867  1    END;

```

00FC 8F BB 00000 LOAD_INDEX_BUCKET:										
				PUSHR	#^M<R2,R3,R4,R5,R6,R7>				1575	
			59 1F	04 02	AA DD 00004	MOVL	4(CTX), BUCKET		1635	
					AA 91 00008	CMPB	2(CTX), #31		1640	
					OD 1F 0000C	BLSSU	1\$			
			00000000G	00 0000000G	8F 01	PUSHL	#CONVS IDX_LIM		1642	
					FB 00014	CALLS	#1, LIB\$STOP			
			50	3A	AA 3C 0001B	1\$:	MOVZWL	58(CTX), R0		
				51	38	AA 3C 0001F	MOVZWL	56(CTX), R1		
			50	51	CO 00023	ADDL2	R1, R0			
					ED 00026	CMPZV	#0, #16, 42(CTX), R0		1663	
			50	44	1F 0002C	BLSSU	3\$			
			29	0000G	CF E8 0002E	BLBS	CONV\$GL_FILL, 2\$		1666	
				50	2C AA	3C 00033	MOVZWL	44(CTX), R0		
				51	38	AA 3C 00037	MOVZWL	56(CTX), R1		
				50	51	CO 0003B	ADDL2	R1, R0		
				52	3A	AA 3C 0003E	MOVZWL	58(CTX), R2		
				50	52	CO 00042	ADDL2	R2, SPACE USED IF_RECORD_ADDED		
				51	18	AB 3C 00045	MOVZWL	24(KEY_DESC), R1		
				52	2C	AA 3C 00049	MOVZWL	44(CTX), R2		
				51	52	C2 0004D	SUBL2	R2, R1		
				52	18	AB 3C 00050	MOVZWL	24(KEY_DESC), R2		
				50	52	C2 00054	SUBL2	R2, R0		
				50	51	D1 00057	CMPL	R1, R0		
					15 0005A	BLEQ	3\$			
			34	0000G	CF E9 0005C	2\$:	BLBC	CONV\$GB PROL V3, 5\$		
				50	A4	AA 9E 00061	MOVAB	-92(R10), CTX M1		
				51	60	02 02	EXTZV	#5, #2, (CTX M1), R1		
					03	ED 0006A	CMPZV	#3, #2, 13(BUCKET), R1		
					23	13 00070	BEQL	5\$		
			0A	5A	5C	AA 9E 00072	3\$:	MOVAB	92(R10), CTX	
				6A	02	E0 00076	BBS	#2, (CTX), 4\$		
				7E	06	AB 9A 0007A	MOVZBL	6(KEY_DESC), -(SP)		
					0000G	30 0007E	BSBW	CONV\$GET_BUCKET		
				5E	04	CO 00081	ADDL2	#4, SP		
					FF79	30 00084	BSBW	LOAD INDEX BUCKET		
				5A	A4	AA 9E 00087	MOVAB	-92(R10), CTX		
				59	04	AA D0 0008B	MOVL	4(CTX), BUCKET		
					0000G	30 0008F	BSBW	CONV\$SWRITE_BUCKET		
					0000G	30 00092	BSBW	CONV\$INIT_BUCKET		
			34	B6	56	5C AA 9E 00095	5\$:	MOVAB	92(R10), CTX P1	
				34	BA	3A AA 28 00099	MOVC3	58(CTX), @52(CTX), @52(CTX_P1)		
				3A	A6	3A AA B0 000A0	MOVW	58(CTX), 58(CTX_P1)		

CONV\$FSTLD  
VO4-000VAX-11 CONVERT  
LOAD\_INDEX\_BUCKET

K 14

15-Sep-1984 23:49:35  
14-Sep-1984 12:14:00VAX-11 Bliss-32 V4.0-742  
[CONV.SRC]CONV\$FSTLD.B32;1Page 47  
(9)

50	38	6A	0C	0000G	CF	E9 000A5	BLBC	CONV\$GB_PROL_V3, 6\$	1748
		A6	02	50	05	EF 000AA	EXTZV	#5, #2, (CTX), R0	1750
					02	A1 000AF	ADDW3	#2, R0, 56(CTX_P1)	
50	38	6A	02	50	0A	11 000B4	BRB	7\$	1752
		A6			05	EF 000B6	6\$: EXTZV	#5, #2, (CTX), R0	
					03	A1 000BB	ADDW3	#3, R0, 56(CTX_P1)	
	03	10	49	0000G	CF	E9 000C0	7\$: BLBC	CONV\$GB_PROL_V3, 10\$	1772
			AB		03	E1 000C5	BBC	#3, 16(REF DESC), 8\$	1778
				0000G	30	000CA	BSBW	CONV\$\$COMPRESS_INDEX	1780
				01	0C	A9 91 000CD	8\$: CMPB	12(BUCKET), #1	1784
					13	12 000D1	BNEQ	9\$	
				50	0000'	CF 3C 000D3	MOVZWL	CONV\$GW_VBN_FS_PTR, R0	1787
						6049 9F 000D8	PUSHAB	(R0)[BUCKET]	
			0000'	CF		9E BO 000DB	MOVW	@(SP)+, SAVE_VBNFS	
			0000'	CF	04	A9 BO 000E0	MOVW	4(BUCKET), SAVE_KEYFRESPC	1788
				50	0000'	CF 3C 000E6	9\$: MOVZWL	CONV\$GW_VBN_FS_PTR, R0	1793
				56	38	AA 3C 000EB	MOVZWL	56(CTX), R6	
						6049 9F 000EF	PUSHAB	(R0)[BUCKET]	
				9E		56 A2 000F2	SUBW2	R6, @(SP)+	1798
						6049 9F 000F5	PUSHAB	(R0)[BUCKET]	
				50		9E 3C 000F8	MOVZWL	@(SP)+, R0	
				51	01	A940 9E 000FB	MOVAB	1(BUCKET)[R0], BKT_CTRL_PTR	
				57	04	A9 3C 00100	MOVZWL	4(BUCKET), BKT_DATA_PTR	1799
				57		59 C0 00104	ADDL2	BUCKET, BKT_DATA_PTR	
	04	A9	04	3A	AA	A0 00107	ADDW2	58(CTX), 4(BUCKET)	
					25	11 0010C	BRB	12\$	1804
				01	0C	A9 91 0010E	10\$: CMPB	12(BUCKET), #1	1803
					06	12 00112	BNEQ	11\$	1815
			0000'	CF	04	A9 BO 00114	MOVW	4(BUCKET), SAVE_FREESPACE	1817
				50	04	A9 3C 0011A	11\$: MOVZWL	4(BUCKET), R0	1824
51		50			59	C1 0011E	ADDL3	BUCKET, R0, BKT_CTRL_PTR	
		56		38	AA	3C 00122	MOVZWL	56(CTX), R6	1825
57		51			56	C1 00126	ADDL3	R6, BKT_CTRL_PTR, BKT_DATA_PTR	
	04	A9	50	3A	56	C0 0012A	ADDL2	R6, R0	1829
	61	30	BA	AA	3A	A1 0012D	ADDW3	58(CTX), R0, 4(BUCKET)	1831
	67	34	BA	AA	56	28 00133	12\$: MOVC3	R6, @48(CTX), (BKT_CTRL_PTR)	1838
		50		3A	AA	28 00138	MOVC3	58(CTX), @52(CTX), -(BKT_DATA_PTR)	1842
		50		3A	AA	3C 0013E	MOVZWL	58(CTX), SPACE_USED	1853
		2A	AA		56	C0 00142	ADDL2	R6, SPACE_USED	
		2C	AA		50	A2 00145	SUBW2	SPACE_USED, 42(CTX)	1855
					50	A0 00149	ADDW2	SPACE_USED, 44(CTX)	1857
				0000G	30	0014D	BSBW	CONV\$WRITÉ_VBN	1863
				00FC	8F	BA 00150	POPR	#^M<R2,R3,R4,R5,R6,R7>	
						05 00154	RSB		1867

; Routine Size: 341 bytes, Routine Base: \_CONV\$FAST\_S + 055B

```
: 1877    1 %SBTTL 'FINISH_INDEX'
: 1878    1 ROUTINE FINISH_INDEX : CL$JSB_REG_9 NOVALUE =
: 1879    1 ++
: 1880    1
: 1881    1 Functional Description:
: 1882    1
: 1883    1     Loads and writes the last buckets in an index.
: 1884    1
: 1885    1 Calling Sequence:
: 1886    1
: 1887    1     FINISH_INDEX()
: 1888    1
: 1889    1 Input Parameters:
: 1890    1     none
: 1891    1
: 1892    1 Implicit Inputs:
: 1893    1     none
: 1894    1
: 1895    1 Output Parameters:
: 1896    1     none
: 1897    1
: 1898    1 Implicit Outputs:
: 1899    1     none
: 1900    1
: 1901    1 Routine Value:
: 1902    1
: 1903    1     CONV$_SUCCESS or error codes
: 1904    1
: 1905    1 Routines Called:
: 1906    1
: 1907    1     CONV$$WRITE_BUCKET
: 1908    1     BACKUP_INDEX
: 1909    1     CONV$$CREATE_HIGH_KEY
: 1910    1     LOAD_INDEX_BUCKET
: 1911    1
: 1912    1 Side Effects:
: 1913    1
: 1914    1     Loads and writes the last buckets in an index. Deallocates memory used
: 1915    1     for bucket buffers.
: 1916    1
: 1917    1 --
: 1918    1
: 1919    1     1909 1
: 1920    1     BEGIN
: 1921    1
: 1922    1     1910 2
: 1923    1     DEFINE_CTX;
: 1924    1     1911 2
: 1925    1     DEFINE_BUCKET;
: 1926    1     1912 2
: 1927    1     DEFINE_KEY_DESC;
: 1928    1
: 1929    1     1913 2
: 1930    1     ! Finish off the data level bucket. The reason why we do this separately
: 1931    1     is that there are no more records to go in this bucket. In the index
: 1932    1     levels there are.
: 1933    1     1914 2
: 1934    1     CTX = .CONV$GL_CTX_BLOCK;
: 1935    1     BUCKET = .CTX [ CTR$L_CURRENT_BUFFER ];
: 1936    1     BUCKET [ BKTSV_LASTBKT ] = _SET;
: 1937    1
: 1938    1
: 1939    1
: 1940    1
: 1941    1
: 1942    1
: 1943    1
: 1944    1
: 1945    1
: 1946    1
: 1947    1
: 1948    1
: 1949    1
: 1950    1
: 1951    1
: 1952    1
: 1953    1
: 1954    1
: 1955    1
: 1956    1
: 1957    1
: 1958    1
: 1959    1
: 1960    1
: 1961    1
: 1962    1
: 1963    1
: 1964    1
: 1965    1
: 1966    1
: 1967    1
: 1968    1
: 1969    1
: 1970    1
: 1971    1
: 1972    1
: 1973    1
: 1974    1
: 1975    1
: 1976    1
: 1977    1
: 1978    1
: 1979    1
: 1980    1
: 1981    1
: 1982    1
: 1983    1
: 1984    1
: 1985    1
: 1986    1
: 1987    1
: 1988    1
: 1989    1
: 1990    1
: 1991    1
: 1992    1
: 1993    1
: 1994    1
: 1995    1
: 1996    1
: 1997    1
: 1998    1
: 1999    1
: 2000    1
: 2001    1
: 2002    1
: 2003    1
: 2004    1
: 2005    1
: 2006    1
: 2007    1
: 2008    1
: 2009    1
: 2010    1
: 2011    1
: 2012    1
: 2013    1
: 2014    1
: 2015    1
: 2016    1
: 2017    1
: 2018    1
: 2019    1
: 2020    1
: 2021    1
: 2022    1
: 2023    1
: 2024    1
: 2025    1
: 2026    1
: 2027    1
: 2028    1
: 2029    1
: 2030    1
: 2031    1
: 2032    1
: 2033    1
: 2034    1
: 2035    1
: 2036    1
: 2037    1
: 2038    1
: 2039    1
: 2040    1
: 2041    1
: 2042    1
: 2043    1
: 2044    1
: 2045    1
: 2046    1
: 2047    1
: 2048    1
: 2049    1
: 2050    1
: 2051    1
: 2052    1
: 2053    1
: 2054    1
: 2055    1
: 2056    1
: 2057    1
: 2058    1
: 2059    1
: 2060    1
: 2061    1
: 2062    1
: 2063    1
: 2064    1
: 2065    1
: 2066    1
: 2067    1
: 2068    1
: 2069    1
: 2070    1
: 2071    1
: 2072    1
: 2073    1
: 2074    1
: 2075    1
: 2076    1
: 2077    1
: 2078    1
: 2079    1
: 2080    1
: 2081    1
: 2082    1
: 2083    1
: 2084    1
: 2085    1
: 2086    1
: 2087    1
: 2088    1
: 2089    1
: 2090    1
: 2091    1
: 2092    1
: 2093    1
: 2094    1
: 2095    1
: 2096    1
: 2097    1
: 2098    1
: 2099    1
: 2100    1
: 2101    1
: 2102    1
: 2103    1
: 2104    1
: 2105    1
: 2106    1
: 2107    1
: 2108    1
: 2109    1
: 2110    1
: 2111    1
: 2112    1
: 2113    1
: 2114    1
: 2115    1
: 2116    1
: 2117    1
: 2118    1
: 2119    1
: 2120    1
: 2121    1
: 2122    1
: 2123    1
: 2124    1
: 2125    1
: 2126    1
: 2127    1
: 2128    1
: 2129    1
: 2130    1
: 2131    1
: 2132    1
: 2133    1
: 2134    1
: 2135    1
: 2136    1
: 2137    1
: 2138    1
: 2139    1
: 2140    1
: 2141    1
: 2142    1
: 2143    1
: 2144    1
: 2145    1
: 2146    1
: 2147    1
: 2148    1
: 2149    1
: 2150    1
: 2151    1
: 2152    1
: 2153    1
: 2154    1
: 2155    1
: 2156    1
: 2157    1
: 2158    1
: 2159    1
: 2160    1
: 2161    1
: 2162    1
: 2163    1
: 2164    1
: 2165    1
: 2166    1
: 2167    1
: 2168    1
: 2169    1
: 2170    1
: 2171    1
: 2172    1
: 2173    1
: 2174    1
: 2175    1
: 2176    1
: 2177    1
: 2178    1
: 2179    1
: 2180    1
: 2181    1
: 2182    1
: 2183    1
: 2184    1
: 2185    1
: 2186    1
: 2187    1
: 2188    1
: 2189    1
: 2190    1
: 2191    1
: 2192    1
: 2193    1
: 2194    1
: 2195    1
: 2196    1
: 2197    1
: 2198    1
: 2199    1
: 2200    1
: 2201    1
: 2202    1
: 2203    1
: 2204    1
: 2205    1
: 2206    1
: 2207    1
: 2208    1
: 2209    1
: 2210    1
: 2211    1
: 2212    1
: 2213    1
: 2214    1
: 2215    1
: 2216    1
: 2217    1
: 2218    1
: 2219    1
: 2220    1
: 2221    1
: 2222    1
: 2223    1
: 2224    1
: 2225    1
: 2226    1
: 2227    1
: 2228    1
: 2229    1
: 2230    1
: 2231    1
: 2232    1
: 2233    1
: 2234    1
: 2235    1
: 2236    1
: 2237    1
: 2238    1
: 2239    1
: 2240    1
: 2241    1
: 2242    1
: 2243    1
: 2244    1
: 2245    1
: 2246    1
: 2247    1
: 2248    1
: 2249    1
: 2250    1
: 2251    1
: 2252    1
: 2253    1
: 2254    1
: 2255    1
: 2256    1
: 2257    1
: 2258    1
: 2259    1
: 2260    1
: 2261    1
: 2262    1
: 2263    1
: 2264    1
: 2265    1
: 2266    1
: 2267    1
: 2268    1
: 2269    1
: 2270    1
: 2271    1
: 2272    1
: 2273    1
: 2274    1
: 2275    1
: 2276    1
: 2277    1
: 2278    1
: 2279    1
: 2280    1
: 2281    1
: 2282    1
: 2283    1
: 2284    1
: 2285    1
: 2286    1
: 2287    1
: 2288    1
: 2289    1
: 2290    1
: 2291    1
: 2292    1
: 2293    1
: 2294    1
: 2295    1
: 2296    1
: 2297    1
: 2298    1
: 2299    1
: 2300    1
: 2301    1
: 2302    1
: 2303    1
: 2304    1
: 2305    1
: 2306    1
: 2307    1
: 2308    1
: 2309    1
: 2310    1
: 2311    1
: 2312    1
: 2313    1
: 2314    1
: 2315    1
: 2316    1
: 2317    1
: 2318    1
: 2319    1
: 2320    1
: 2321    1
: 2322    1
: 2323    1
: 2324    1
: 2325    1
: 2326    1
: 2327    1
: 2328    1
: 2329    1
: 2330    1
: 2331    1
: 2332    1
: 2333    1
: 2334    1
: 2335    1
: 2336    1
: 2337    1
: 2338    1
: 2339    1
: 2340    1
: 2341    1
: 2342    1
: 2343    1
: 2344    1
: 2345    1
: 2346    1
: 2347    1
: 2348    1
: 2349    1
: 2350    1
: 2351    1
: 2352    1
: 2353    1
: 2354    1
: 2355    1
: 2356    1
: 2357    1
: 2358    1
: 2359    1
: 2360    1
: 2361    1
: 2362    1
: 2363    1
: 2364    1
: 2365    1
: 2366    1
: 2367    1
: 2368    1
: 2369    1
: 2370    1
: 2371    1
: 2372    1
: 2373    1
: 2374    1
: 2375    1
: 2376    1
: 2377    1
: 2378    1
: 2379    1
: 2380    1
: 2381    1
: 2382    1
: 2383    1
: 2384    1
: 2385    1
: 2386    1
: 2387    1
: 2388    1
: 2389    1
: 2390    1
: 2391    1
: 2392    1
: 2393    1
: 2394    1
: 2395    1
: 2396    1
: 2397    1
: 2398    1
: 2399    1
: 2400    1
: 2401    1
: 2402    1
: 2403    1
: 2404    1
: 2405    1
: 2406    1
: 2407    1
: 2408    1
: 2409    1
: 2410    1
: 2411    1
: 2412    1
: 2413    1
: 2414    1
: 2415    1
: 2416    1
: 2417    1
: 2418    1
: 2419    1
: 2420    1
: 2421    1
: 2422    1
: 2423    1
: 2424    1
: 2425    1
: 2426    1
: 2427    1
: 2428    1
: 2429    1
: 2430    1
: 2431    1
: 2432    1
: 2433    1
: 2434    1
: 2435    1
: 2436    1
: 2437    1
: 2438    1
: 2439    1
: 2440    1
: 2441    1
: 2442    1
: 2443    1
: 2444    1
: 2445    1
: 2446    1
: 2447    1
: 2448    1
: 2449    1
: 2450    1
: 2451    1
: 2452    1
: 2453    1
: 2454    1
: 2455    1
: 2456    1
: 2457    1
: 2458    1
: 2459    1
: 2460    1
: 2461    1
: 2462    1
: 2463    1
: 2464    1
: 2465    1
: 2466    1
: 2467    1
: 2468    1
: 2469    1
: 2470    1
: 2471    1
: 2472    1
: 2473    1
: 2474    1
: 2475    1
: 2476    1
: 2477    1
: 2478    1
: 2479    1
: 2480    1
: 2481    1
: 2482    1
: 2483    1
: 2484    1
: 2485    1
: 2486    1
: 2487    1
: 2488    1
: 2489    1
: 2490    1
: 2491    1
: 2492    1
: 2493    1
: 2494    1
: 2495    1
: 2496    1
: 2497    1
: 2498    1
: 2499    1
: 2500    1
: 2501    1
: 2502    1
: 2503    1
: 2504    1
: 2505    1
: 2506    1
: 2507    1
: 2508    1
: 2509    1
: 2510    1
: 2511    1
: 2512    1
: 2513    1
: 2514    1
: 2515    1
: 2516    1
: 2517    1
: 2518    1
: 2519    1
: 2520    1
: 2521    1
: 2522    1
: 2523    1
: 2524    1
: 2525    1
: 2526    1
: 2527    1
: 2528    1
: 2529    1
: 2530    1
: 2531    1
: 2532    1
: 2533    1
: 2534    1
: 2535    1
: 2536    1
: 2537    1
: 2538    1
: 2539    1
: 2540    1
: 2541    1
: 2542    1
: 2543    1
: 2544    1
: 2545    1
: 2546    1
: 2547    1
: 2548    1
: 2549    1
: 2550    1
: 2551    1
: 2552    1
: 2553    1
: 2554    1
: 2555    1
: 2556    1
: 2557    1
: 2558    1
: 2559    1
: 2560    1
: 2561    1
: 2562    1
: 2563    1
: 2564    1
: 2565    1
: 2566    1
: 2567    1
: 2568    1
: 2569    1
: 2570    1
: 2571    1
: 2572    1
: 2573    1
: 2574    1
: 2575    1
: 2576    1
: 2577    1
: 2578    1
: 2579    1
: 2580    1
: 2581    1
: 2582    1
: 2583    1
: 2584    1
: 2585    1
: 2586    1
: 2587    1
: 2588    1
: 2589    1
: 2590    1
: 2591    1
: 2592    1
: 2593    1
: 2594    1
: 2595    1
: 2596    1
: 2597    1
: 25
```

```
1934      1925 2    | Write the data level bucket
1935      1926 2
1936      1927 2
1937      1928 2
1938      1929 2    | If the last data bucket was a continuation bucket then backup one
1939      1930 2    index record and put the high key there
1940      1931 2
1941      1932 2    IF .CONTINUATION
1942      1933 2    THEN
1943      1934 2    BACKUP_INDEX();
1944      1935 2
1945      1936 2    | Create the high key index record to finish things off
1946      1937 2
1947      1938 2    CONV$$CREATE_HIGH_KEY();
1948      1939 2
1949      1940 2    | Write the last index records into the buckets and then write the
1950      1941 2    buckets out
1951      1942 2
1952      1943 2    Move up to level 1
1953      1944 2
1954      1945 2    CTX = .CTX + CTX$K_BLN;
1955      1946 2
1956      1947 2    | Loop until each level is processed
1957      1948 2
1958      1949 2    WHILE .CTX [ CTX$V_RDY ]
1959      1950 2    DO
1960      1951 2    BEGIN
1961      1952 2
1962      1953 2    LOCAL   CTX_P1 : REF BLOCK [ ,BYTE ];
1963      1954 2
1964      1955 2    | This call to load bucket will finish off this level bucket and create
1965      1956 2    the index to the next.
1966      1957 2
1967      1958 2    LOAD_INDEX_BUCKET();
1968      1959 2
1969      1960 2    | Before we write out the last bucket set some control info. in it
1970      1961 2
1971      1962 2    BUCKET = .CTX [ CTXSL_CURRENT_BUFFER ];
1972      1963 2    BUCKET [ BKT$V_LASTBKT ] = _SET;
1973      1964 2
1974      1965 2    CTX_P1 = .CTX + CTX$K_BLN;
1975      1966 2
1976      1967 2    | If there is no bucket above this one then this is the root
1977      1968 2
1978      1969 2    IF ( NOT .CTX_P1 [ CTX$V_RDY ] )
1979      1970 2    THEN
1980      1971 2    BEGIN
1981      1972 2    BUCKET [ BKT$V_ROOTBKT ] = SET;
1982      1973 2    KEY_DESC [ KEY$B_ROOTLEV ] = .CTX [ CTX$B_LEVEL ];
1983      1974 2    KEY_DESC [ KEY$L_ROOTVBN ] = .CTX [ CTX$L_CURRENT_VBN ];
1984      1975 2    KEY_DESC [ KEYSV_INITIDX ] = _CLEAR
1985      1976 2    END;
1986      1977 2
1987      1978 2    | Write the last bucket at this level
1988      1979 2
1989      1980 2    CONV$$WRITE_BUCKET();
1990      1981 2
```

```

: 1991    1982 3      ! Clear the bucket ready flag at this level
: 1992    1983 3
: 1993    1984 3
: 1994    1985 3      CTX [ CTX$V_RDY ] = _CLEAR;
: 1995    1986 3
: 1996    1987 3      ! Prepare to work on the bucket one level up
: 1997    1988 3      CTX = .CTX_P1
: 1998    1989 3
: 1999    1990 2      END;
: 2000    1991 2
: 2001    1992 2      ! Make sure the last IO has completed
: 2002    1993 2
: 2003    1994 2      SWAIT( RAB=CONV$AB_OUT_RAB );
: 2004    1995 2
: 2005    1996 2      ! Any more IO will be Synchronous
: 2006    1997 2
: 2007    1998 2      CONV$AB_OUT_RAB [ RAB$V_ASY ] = _CLEAR;
: 2008    1999 2
: 2009    2000 2      RETURN
: 2010    2001 2
: 2011    2002 1      END;

```

## .EXTRN SYSSWAIT

			52 DD 00000 FINISH_INDEX:			
			5A 0000' CF D0 00002	PUSHL	R2	1869
		OD	59 04 AA D0 00007	MOVL	CONV\$GL_CTX_BLOCK, CTX	1921
			A9 01 88 0000B	MOVL	4(CTX), BUCRET	1922
			0000G 30 0000F	BISB2	#1, 13(BUCKET)	1923
			03 0000' CF E9 00012	BSBW	CONV\$SWRITE_BUCKET	1927
			0000V 30 00017	BLBC	CONTINUATION, 1\$	1932
			0000G 30 0001A 1\$:	BSBW	BACKUP INDEX	1934
			5C AA 9E 0001D	MOVAB	CONV\$CREATE_HIGH_KEY	1938
			6A 02 E1 00021 2\$:	BBC	92(R10), CTX	1945
		30	FE83 30 00025	BSBW	#2, (CTX), 4\$	1949
			59 04 AA D0 00028	MOVL	LOAD_INDEX_BUCKET	1958
			A9 01 88 0002C	BISB2	4(CTX), BUCKET	1962
			52 5C AA 9E 00030	MOVAB	#1, 13(BUCKET)	1963
		12	62 02 E0 00034	BBS	92(R10), CTX_P1	1965
			OD A9 02 88 00038	BISB2	#2, (CTX_P1), 3\$	1969
			09 AB 02 AA 90 0003C	MOVB	2(CTX), 9(KEY DESC)	1972
			OC AB 08 AA D0 00041	MOVL	8(CTX), 12(KEY DESC)	1973
			10 AB 10 8A 00046	BICB2	#16, 16(KEY DESC)	1974
			0000G 30 0004A 3\$:	BSBW	CONV\$SWRITE_BUCKET	1980
			6A 04 8A 0004D	BICB2	#4, (CTX)	1984
			5A 52 D0 00050	MOVL	CTX_P1, CTX	1988
			CC 11 00053	BRB	2\$	
			0000G CF 9F 00055 4\$:	PUSHAB	CONV\$AB_OUT_RAB	1994
		00000000G	00 01 FB 00059	CALLS	#1, SYSSWAIT	
		0000G CF	01 8A 00060	BICB2	#1, CONV\$AB_OUT_RAB+4	1998
			04 BA 00065	POPR	#^M<R2>	2002
			05 00067	RSB		

; Routine Size: 104 bytes, Routine Base: \_CONV\$FAST\_S + 06B0

CONVFSTLD  
V04-000

VAX-11 CONVERT  
FINISH\_INDEX

B 15  
15-Sep-1984 23:49:35    VAX-11 Bliss-32 v4.0-742  
14-Sep-1984 12:14:00    [CONV.SRC]CONVFSTLD.B32;1

Page 51  
(10)

```
2013    2003 1 %SBTTL 'BACKUP INDEX'
2014    2004 1 ROUTINE BACKUP_INDEX : CL$JSB_REG_9 NOVALUE =
2015    2005 1 ++
2016    2006 1
2017    2007 1     Functional Description:
2018    2008 1
2019    2009 1     Calling Sequence:
2020    2010 1         BACKUP_INDEX()
2021    2011 1
2022    2012 1
2023    2013 1     Input Parameters:
2024    2014 1         none
2025    2015 1
2026    2016 1     Implicit Inputs:
2027    2017 1         none
2028    2018 1
2029    2019 1     Output Parameters:
2030    2020 1         none
2031    2021 1
2032    2022 1     Implicit Outputs:
2033    2023 1         none
2034    2024 1
2035    2025 1     Routine Value:
2036    2026 1         none
2037    2027 1
2038    2028 1     Routines Called:
2039    2029 1         none
2040    2030 1
2041    2031 1
2042    2032 1
2043    2033 1     Side Effects:
2044    2034 1         Loads and writes the last buckets in an index. Deallocates memory used
2045    2035 1         for bucket buffers.
2046    2036 1
2047    2037 1
2048    2038 2     BEGIN
2049    2039 2
2050    2040 2     DEFINE_CTX;
2051    2041 2     DEFINE_BUCKET;
2052    2042 2     DEFINE_KEY_DESC;
2053    2043 2
2054    2044 2     LOCAL
2055    2045 2         VBN_SIZE,
2056    2046 2         CTX_P1      : REF BLOCK [ ,BYTE ],
2057    2047 2         RECORD_CTRL   : REF BLOCK [ ,BYTE ];
2058    2048 2
2059    2049 2         CTX_P1 = .CTX + CTX$K_BLN;
2060    2050 2
2061    2051 2         BUCKET = .CTX_P1 [ CTX$L_CURRENT_BUFFER ];
2062    2052 2
2063    2053 2         | If the last data bucket was a continuation bucket then we will be backing
2064    2054 2         | up index record which requires using the vbn in the last record. We
2065    2055 2         | can fake out conv$Swrite_vbn (called in conv$Screate_high_key) by stuffing
2066    2056 2         | the vbn in the ctx field. This is ok since it it never referenced again.
2067    2057 2
2068    2058 2         | Get the size of the vbn in the old record (in bits)
2069    2059 2
```

```

2070    2060 2     VBN_SIZE = ( .CTX_P1 [ CTX$V_VBN ] + 2 ) * 8;
2071    2061 2
2072    2062 2     ! Backup the pointers in the bucket above and get the vbn in the record
2073    2063 2
2074    2064 2     IF .CONV$GB_PROL_V3
2075    2065 2     THEN
2076    2066 3     BEGIN
2077    2067 3
2078    2068 3     ! For prologue 3 the vbn is at where we are (they go backwards)
2079    2069 3
2080    2070 3     RECORD_CTRL = .BUCKET [ BKT$W_VBNFS ] + .BUCKET + 1;
2081    2071 3
2082    2072 3     CTX [ CTX$L_CURRENT_VBN ] = .RECORD_CTRL [ 0,0,.VBN_SIZE,0 ];
2083    2073 3
2084    2074 3     BUCKET [ BKT$W_VBNFS ] = .SAVE_VBNFS;
2085    2075 3     BUCKET [ BKT$W_KEYFRESPC ] = .SAVE_KEYFRESPC
2086    2076 3
2087    2077 3     END
2088    2078 2     ELSE
2089    2079 3     BEGIN
2090    2080 3
2091    2081 3     BUCKET [ BKT$W_FREESPACE ] = .SAVE_FREESPACE;
2092    2082 3
2093    2083 3     RECORD_CTRL = .BUCKET [ BKT$W_FREESPACE ] + .BUCKET;
2094    2084 3
2095    2085 3     CTX [ CTX$L_CURRENT_VBN ] = .RECORD_CTRL [ 1,0,.VBN_SIZE,0 ]
2096    2086 3
2097    2087 2     END;
2098    2088 2
2099    2089 2     RETURN
2100    2090 2
2101    2091 1     END;

```

52 DD 00000 BACKUP_INDEX:									
				PUSHL	R2				2004
50	50	5C	AA	9E 00002	MOVAB	92(R10), CTX_P1			2049
	59	04	A0	D0 00006	MOVL	4(CTX_P1), BUCKET			2051
	02		05	EF 0000A	EXTZV	#5, #2, (CTX_P1), R0			2060
	50		08	C4 0000F	MULL2	#8, VBN_SIZE			
	50		10	C0 00012	ADDL2	#16, VBN_SIZE			
	26	0000G	CF	E9 00015	BLBC	CONV\$GB_PROL_V3, 1\$			2064
	52	0000'	CF	3C 0001A	MOVZWL	CONV\$GW_VBN_FS_PTR, R2			2070
			6249	9F 0001F	PUSHAB	(R2)[BUCKET]			
08 AA	51		9E	3C 00022	MOVZWL	a(SP)+, R1			
	51		01 A941	9E 00025	MOVAB	1(BUCKET)[R1], RECORD_CTRL			
	50		00	EF 0002A	EXTZV	#0, VBN_SIZE, (RECORD_CTRL), 8(CTX)			2072
			6249	9F 00030	PUSHAB	(R2)[BUCKET]			2074
	9E	0000'	CF	B0 00033	MOVW	SAVE_VBNFS, a(SP)+			
	04	A9	0000'	CF B0 00038	MOVW	SAVE_KEYFRESPC, 4(BUCKET)			2075
	04	A9	0000'	14 11 0003E	BRB	2\$			
	51	04	A9	3C 00040 1\$:	MOVW	SAVE_FREESPACE, 4(BUCKET)			2081
	51		59	C0 00046	MOVZWL	4(BUCKET), RECORD_CTRL			2083
				59 C0 0004A	ADDL2	BUCKET, RECORD_CTRL			

CONV\$FSTLD  
V04-000 VAX-11 CONVERT  
BACKUP\_INDEX

E 15  
15-Sep-1984 23:49:35 VAX-11 Bliss-32 v4.0-742  
14-Sep-1984 12:14:00 [CONV.SRC]CONVFSTLD.L32;1

Page 54  
(11)

08 AA 01 A1 50 00 EF 0004D EXTZV #0, VBN\_SIZE, 1(RECORD\_CTRL), 8(CTX)  
04 BA 00054 2\$: POPR #^M<R2>  
05 00056 RSB

; 2085  
; 2091

; Routine Size: 87 bytes, Routine Base: \_CONV\$FAST\_S + 0718

; 2102 2092 1  
; 2103 2093 0 END ELUDOM

.EXTRN LIB\$STOP

PSECT SUMMARY

Name	Bytes	Attributes
_CONV\$FAST_D	28	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
_CONV\$FAST_S	1903	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(2)

Library Statistics

File	Total	Symbols	Pages	Processing
	Loaded	Percent	Mapped	Time
\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	56	0	00:01.8
\$255\$DUA28:[CONV.SRC]CONVERT.L32;1	165	43	26	00:00.2

; Information: 1  
; Warnings: 0  
; Errors: 0

COMMAND QUALIFIERS

; BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:CONVFSTLD/OBJ=OBJ\$:CONVFSTLD MSRC\$:CONVFSTLD/UPDATE=(ENH\$:CONVFSTLD)

; Size: 1903 code + 28 data bytes  
; Run Time: 00:43.9  
; Elapsed Time: 02:19.3  
; Lines/CPU Min: 2863  
; Lexemes/CPU-Min: 16797  
; Memory Used: 250 pages  
; Compilation Complete

0065 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY

CONVDATA  
LIS

CONVSTD  
LIS

CONVFSTIO  
LIS

CONVFSTRC  
LIS

CONVFILES  
LIS

CONVERRUR  
LIS

CONVFASTM  
LIS

CONVDCL  
LIS